PREDICTING THE OBJECT-ORIENTED CLASS STABILITY USING SOFTWARE METRICS

BY

Yagoub Mohammad Abdullah Eisa

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

May 2012

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **YAGOUB MOHAMMAD ABDULLAH EISA** under the direction of his thesis advisors and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER SCIENCE.**

Thesis Committee

Dr. M. Alshayeb (Chairman)

Dr. M. Ahmed (Member)

Dr. M. Niazi (Member)

Dr. Adel F. Ahmed
Department Chairman

Dr. Salam A. Zummo
Dean of Graduate Studies

3/6/12
Date

DEDICATION

To my affectionate mother, for her constant prayers

To my precious wife, who has always supported me and has

been a great source of motivation and inspiration.

ACKNOWLEDGMENT

In the name of Allah, most gracious, most merciful

I would like to express my deep appreciation for the opportunity and encouragement extended to me by my employer Saudi Aramco, the world leading oil and Gas Company to pursue a Masters degree in Computer Science.

I further extend my acknowledgment to my thesis advisor Dr. Mohammad Alshayeb for his priceless support, sound advice, and guidance that have been tremendously helpful throughout the research. Also, I owe an immense debt of gratitude to committee members Dr. Moataz Ahmed and Dr. Mahmoud Niazi for their excellent support and contribution. An appreciation is attributed to KFUPM for establishing an innovative platform for researchers to explore and excel.

Special thanks go to my beloved family, brothers, sisters, and children for their continuous support and encouragement. Lastly, I convey my deep gratitude to colleagues and friends who provided invaluable assistance throughout the research

.

TABLE OF CONTENETS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Name: Yagoub Mohammad Abdullah Eisa

Title: Predicting The Object-Oriented Class Stability Using Software Metrics

Major Field: Computer Science

Date of Degree: May 2012

Class stability in object-oriented systems is an important factor for software quality. Therefore, many class stability metrics have been proposed to measure class stability. Most of the class stability metrics have not been investigated or involved in any prediction studies. Hence, such studies help in improving the software quality and can be used as an early quality indicator of class stability/instability in order to help in reducing maintenance cost and efforts. This thesis seeks to investigate whether the class stability metrics can be used as a predictor for software quality using software metrics. Therefore, prediction models are created to predict the object-oriented class stability. Two artificial intelligence techniques are used, Neural Network and Support Vector Machine, in addition to widely used prediction techniques, Multi Linear Regression and Logistic Regression. Software metrics concerning coupling, cohesion, and complexity are used as predictor variables, where the output of the class stability metrics is used as a target variable of the prediction model.

ملخص الرسالة

الاســـــــــم: يعقـوب بن محمـد بن عبـدالله عيسـى

عنوان الرسالة: توقع استقرار اصناف البرمجة الموجهة باستخدام مقاييس البرمجيات

التخـصـــــص: علوم الحاسب الالي

تاريخ التخـرج: مايو 2012

يمثل استقرار الأصناف في البرمجة الموجهة عامل اساسي للمحافظة على جودة البرمجيات. وبالتالي، تم تطوير العديد من المقاييس لقياس درجة استقرار اصناف البرمجة الموجهة. بالرغم من ذلك، فان معظم مقاييس الاستقرار لم تخضع لأي دراسة متعلقة بتوقع استقرار الأصناف. ومن ثم، فان مثل هذه الدراسات تساعد في تحسين جودة البرامجيات، ويمكن استخدامها كمؤشر جودة من أجل المساعدة في الحد منتكاليف صيانة البرمجيات. تسعى هذه الرسالة الى دراسة امكانية استخدام مقاييس الاستقرار كمؤشر للجودة عن طريق توقع استقرار اصناف البرمجة الموجهة.

CHAPTER 1

INTRODUCTION

Software quality is an important topic of research for software engineers. Producing software that eliminates defect and satisfies user requirements is an objective of software quality. Key factors such as stability, maintainability, reliability, efficiency, etc. are used to assess the quality of the software. Software quality can be classified into internal and external quality characteristics [1]. "*External quality characteristics are those parts of a product that face its users, whereas internal quality characteristics are those that do not".* Internal quality characteristics are not visible to the end-user, but they are important to the project management team and developers as they are the source of information for evaluating the project plan and progress.

Stability is one of the important internal quality characteristics that assist in reducing the maintenance cost and efforts. Building a software and keeping it stable is a

major challenge as the software keeps on changing due to new user requirements and new technological advances whereas unstable software leads to more effort and high maintenance cost. Furthermore, classes being the basic components of software are also subjected to frequent modifications. Therefore, stable classes may contribute to reducing maintenance cost and efforts [2]. As a result, many class stability metrics have been proposed by researchers to measure and assess the class attributes in order to improve the software quality and reduce maintenance cost and efforts [2-5]. Each metric uses different quality factor(s) to measure the stability of the class and most of them have not been investigated or involved in any prediction studies. Therefore, there is a need to investigate the proposed class stability metrics using prediction models to assess each metric and evaluate its reliability of ranking classes according to their stability quality. A prediction model is built based upon the relationship between quality factors and software metrics. However, this relationship is complex and can limit the accuracy of the models [6]. As a result, artificial intelligence (AI) techniques can be utilized for assessing the quality of the software and verifying the suitability of the models using existing software metrics. In this research, prediction models are created using artificial intelligence (AI) and statistical techniques to predict class stability using software metrics. This chapter discusses the problem statement, the significance of the problem, the research objective and its contributions, and finally the thesis organization.

1.1 Problem Statement

Software stability is important as a quality factor that assists in reducing the cost and efforts of software maintenance. For this reason, many class stability metrics have been proposed by researchers using different quality factors to measure software stability. Li et al. [3] proposed class implementation instability (CII) which is based on the changes in lines of code (LOC). Grosser et al. [5] proposed class stability metric which used class interface or public and protected methods to measure the class stability. Daniel et al. [4] proposed class stability metric with respect to number of methods (NOM). Recently, Alshayeb et al. [2] identified properties that affect class stability and then used them to propose a new class stability metric (CSM). The proposed class stability metrics are used to measure stability of the existing versions. As developers or project management team, it is important to predict the stability of the future versions in order to make the right decision. So, can stability be predicted using other software metrics? Which metric can give the most acceptable stability results? Can these metrics be used as a predictor for software quality? Can they be used as early quality indicators? This research seeks answers to these questions.

1.2 Rationale: Problem Importance

Stability of the class in an object-oriented system is an important contributing factor for software quality. Therefore, many class stability metrics have been proposed by researchers to measure class stability and support software quality [2-5]. Building and keeping software stable is a big challenge as adapting the software to new requirements are necessary because of changing user needs and technological advances, whereas the use of unstable software may lead to high maintenance cost and efforts. Therefore, it is important to manage the stability of the class along different versions. As a result, prediction techniques have been widely used as a quality indicator in the early development stage of software. Grosser et al. [5] proposed an approach to explore structural similarities between classes to predict their chances of becoming unstable. Case-Based Reasoning (CBR) technique was used to build a prediction model for class stability. To predict instability of a class using this approach, similar cases that have occurred in the past need to exist. Bouktif et al. [7] proposed two genetic algorithm approaches that combines/adapts a set of existing decision tree models to predict software stability. Azar and Korkmaz [8] proposed a hybrid heuristic approach that is based on genetic algorithm, simulated annealing, and tabu search to recombine existing rule-based prediction models to predict class stability. Most of the class stability metrics have not been investigated or involved in any prediction studies. Hence, such studies help in improving the software quality and can be used as an early quality indicator of class stability/instability in order to help in reducing maintenance cost and efforts. Therefore,

an in depth study is required to predict the object-oriented class stability using different software metrics and different prediction techniques.

1.3 Research Objective & Contributions

Predicting class stability using software metrics improves the software quality and reduces the maintenance cost and effort. Thus the following research objective is set:

Predicting the object-oriented class stability using software metrics

The main objective is further divided into sub-objectives:

1. Identify the software metrics that can be used to build the prediction models.
2. Quantitatively assess the relationship between the selected software metrics and the stability that is measured by the class stability metrics.
3. Build artificial intelligence prediction models, using software metrics, to predict class stability as measured by the class stability metrics.
4. Build statistical prediction models, using software metrics, to predict class stability as measured by the class stability metrics.
5. Analyze and compare the performance of the different prediction models to evaluate their accuracy.
6. Predict class stability using software metrics.

In a nutshell the research contribution can be summarized as building prediction models for the existing object-oriented class stability using software metrics.

1.4 Thesis Organization

This thesis is organized into seven chapters: Introduction, Literature Review, Background, Data Collection and Analysis, Regression Experiment, Classification Experiment, and finally Conclusion and Future Work. This section provides a summary of the remaining chapters.

Chapter Two – Literature Review: The chapter presents a comprehensive review of different stability level, different measurement types, and the artificial intelligence techniques that are used to predict software quality. It also introduces the contributions of various researchers in this field and the methodology used by each of them.

Chapter Three – Background: The chapter describes the key topics related to this thesis. It provides detailed information on stability and its definitions, class stability metrics and how they are used to measure stability, the importance of measurement validation, and different types of stability validation.

Chapter Four – Data Collection and Analysis: The chapter describes in detail the process of collecting and analyzing the data that is used to conduct the experiments, how it is ordered and organized in order to extract useful information. In addition, software tools that are used to extract metrics are also introduced in this chapter.

Chapter Five – Regression Experiments: The chapter describes the proposed hypotheses, regression tools, and experimental setup. It also introduces the results of the prediction models of neural network, support vector machine, and multiple linear regressions. Finally, it discusses the results, observations and the comparisons between regression models.

Chapter Six – Classification Experiments: The chapter describes the proposed hypotheses, classification tools, and experimental setup. It also introduces the results of classification prediction models of neural network, support vector machine, and logistic regression. Finally, it discusses the results, observations and the comparisons between classification models.

Chapter Seven – Conclusion and Future Work: The final chapter presents the thesis conclusion that drawn from all the pervious chapters, threats to validity, and discusses the possible directions for future research.

CHAPTER 2

LITERATURE REVIEW

This chapter presents the literature review of the previous research studies on software stability and artificial Intelligence techniques for software quality prediction.

2.1 Class Stability

Object-oriented class forms the basic components of the software architecture and it can be an important source of information to verify the project plan. Therefore, many metrics were proposed by researchers to measure class stability. Class stability metrics can be used as early indicator for any out-of-control situation that may arise during the maintenance [9]. Li et al. [3] performed an empirical study to evaluate object-oriented system evolution. Three metrics were proposed, two metrics for system instability and one for class implementation instability (CII). CII metric was designed based on the

changes in lines of code (LOC). Grosser et al. [5] proposed a class stability metric that is based on class interface or public and protected methods. This metric was used as part of the Case-Based Reasoning (CBR) to build the prediction model. Stab class stability metric was proposed by Daniel et al. [4] which measures class stability with respect to number of methods (NOM) of a class. Alshayeb et al. [2] identified properties that affect class stability and then used them to propose new class stability metrics (CSM). Considering many factors may improve stability measurement and reduce the maintenance cost and effort, and yield more accurate and informative assessment of class stability. The class stability metrics are discussed in more detail in the background chapter.

2.2 System Stability

System stability is another level of software stability. Li et al. [3] proposed three stability metrics, System Design Instability (SDI), System Implementation Instability (SII), and Class Implementation Instability (CII) to measure changes in the system design and its implementation. Alshayeb and Li [10] revised the System Design Instability (SDI) metric and performed an empirical study on two systems developed using an agile process. Fayad [11-13] introduced two new concepts“Enduring Business Themes” (EBTs) and “Business Objects” (BOs) to solve the project reengineering issue and to produce a stable software product. Yau and Collofello [14] presented measures for design stability which indicated the modification effects on the program at the design level. Soong [15] used connectivity matrix and random Markovian process to measure program stability.

2.3 Architecture Stability

Another level of software stability is the architecture stability, it is defined as the amount of architecture that remains intact while incorporating changes at the same time made because of evolution [16]. Bansiya [17] used software metrics to propose an approach to evaluate framework architectural stability. Ahmed et al. [18] proposed two similarity metrics, Shallow Semantic Similarity Metric (SSSM) and Relationship-Based Similarity Metric (RBSM) to measure the architectural stability of an object-oriented system. Subrina et al. [19] combined retrospective and predictive analysis techniques and proposed a metric-based approach to evaluate the architectural stability of a software system.

2.4 Measurement Validation

The objective of this research is to empirically validate object-oriented class stability metrics. Validation assures that the measurements assess what they are supposed to measure. The validation is of two types, theoretical and empirical validations. Theoretical validation is the process of making certain that the measure is a proper numerical characterization of a claimed attribute. It can be done by satisfying the representation condition or by correlating the measure with some well-known existent metric [20]. Empirical validation is the process of establishing the accuracy of a prediction system by comparing the model performance with known data in a given environment [21, 22]. An empirical validation usually includes experimentation and hypothesis testing [23-25]. As stability is an important software quality factor, empirical

validation of class stability metrics using perdition models can be used as early indicators of software quality.

2.5 Artificial Intelligence techniques for software quality prediction

Software quality is the degree to which a system, component, or process meets the specified requirements. There are several ways to ensure the quality of the software and prediction is one of the techniques that can be used to assure software quality. Artificial Intelligence (AI) techniques try to imitate the characteristics that are associated with intelligence in human behavior to solve a given problem. This inspired the researchers to predict software quality using AI techniques such as neural network, support vector machine, genetic algorithm, etc. This section introduces different research studies that used AI techniques to predict software quality.

2.5.1 Neural Network

Neural network is an artificial Intelligence technique that is designed in a way of learning mechanism which works in a similar fashion as a human brain [26]. It consists of a large number of neurons which are connected to each other by direct links associated with weights. It can be used to build a predictive model as it is capable of modeling complex functions [6].

- **Zhong et al. [27]** proposed a prediction model based on Wavelet Neural Network (WNN) to predict software reliability using software metrics. WNN introduces two

new parameters; the scale (or frequency) factor and the translation factor that operates on different frequency components, and then evaluates each component with a resolution matched to its scale. The WNN function replaces the sigmoid function in the Back-Propagation neural network. The WNN layers constructed as follows, input layer:

$$\boldsymbol{net_{ni}} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{j}} \boldsymbol{w_{ih} x_{ni}} \qquad \textbf{(1)}$$

Where n is the number of input samples, i and j are the number of neurons in input and hidden layers respectively, (w_{ih}) is the connection weight between neurons in input and hidden layers, (x_{ni}) is the value of the input variable, finally (net_{ni}) is the output of the input layer's samples. The hidden layer constructed as follows:

$$\emptyset\, \boldsymbol{a_h}, \boldsymbol{b_h}\, (\boldsymbol{net_{ni}}) = \emptyset\, (\frac{\boldsymbol{net_{ni}} - \boldsymbol{b_h}}{\boldsymbol{a_h}}) \qquad \textbf{(2)}$$

Where (a_h) is the scale factor, (b_h) is the translation factor, (Φ a_h, $b_h(net_{ni})$) is the output of the hidden layer neurons of the input samples. Finally the output layer constructed as follows:

$$\boldsymbol{y(t)_n} = \boldsymbol{\sigma}\, (\sum_{\boldsymbol{h=1}}^{\boldsymbol{k}} \boldsymbol{w_h}\, \emptyset\, \boldsymbol{a_h}, \boldsymbol{b_h}\, (\boldsymbol{net_{ni}})) \qquad \textbf{(3)}$$

Where (w_h) is the connection weight between neurons in the hidden and the output layers, and ($y(t)_n$) is the actual output of the input samples, where the ($\dot{y}(t)_n$) is the

expected output of the samples. WNN updated its parameters using the sum squared error E_n of the actual and expected output values as follows:

$$E_n = \frac{1}{2}\sum_{k=1}^{n}(Y(t)_k - \dot{\hat{y}}(t)_k)^2 \qquad (4)$$

WNN applied genetic algorithm (GA) technique to optimize its parameters and to obtain the initial weight of WNN neurons. WNN models outperform the Back-Propagation neural network and Generalized Regression neural network [27].

- **Thwin and Quah [6]** used Ward Neural Network (WNN) and Generalized Regression Neural Network (GRNN) models to predict the number of defects and the lines changed per class using software metrics.

Ward Neural Network (WNN)

WNN is a Back-Propagation neural network using different activation functions in each slab of the hidden layer as shown in Figure 2-1. Slab is a group of neurons; Slab 1 and Slab 5 represent the input and the output layers respectively, where as Slab 2 to 4 represent the hidden layers. Tangent, Gaussian, and Gaussian Complement activation functions are used in the Slab of the hidden layer to offer different ways to view the data. Linear function is used in output Slab.

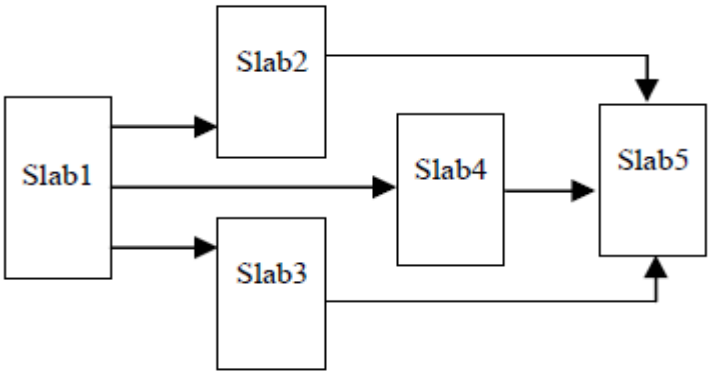


Figure 2-1: WNN Architecture

Generalized Regression Neural Network (GRNN)

GRNN is a memory-based neural network that can deal with continuous variables and sparse data effectively as shown Figure 2-2. It has three Slabs and the number of patterns in the training set determines the number of neurons in the hidden layer. GRNN is a one-pass learning algorithm with highly parallel structure. The learning rate and momentum parameters are not required, instead there is adjustable parameter called smoothing factor which is applied after the network is trained. GRNN models predict more accurately than the Ward network models.

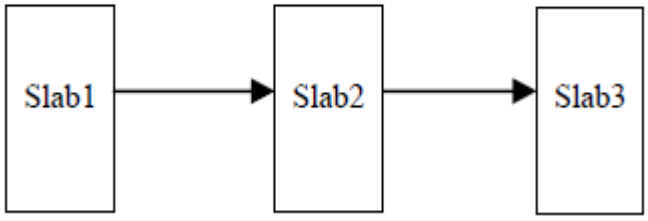


Figure 2-2: GRNN Architecture

- **Kanmani et al. [28]** proposed classification prediction models based on Back-Propagation Neural Network (BPNN) and Probabilistic Neural Network (PNN) to predict fault proneness using software metrics. Four quality parameters were used to evaluate the goodness of the classification models; misclassification, correctness, completeness, and effectiveness. They found that BPNN and PNN models perform better than statistical models such as logistic regression and among the two neural network models, PNN outperformed Benin predicting the fault proneness of the software.

- Peng et al. [29] builds a prediction model based on Fuzzy Neural Network (FNN). The model is proposed to determine the cause-and-effect of relationship between software metrics and quality factors. FNN combines the self-learning capability of neural network and expresses fuzzy information capability of fuzzy logic. Al-Jamimi and Ghouti [30] performed a study to evaluate the performance of fault predictor models based on Probabilistic Neural Network (PNN) on five different public NASA databases. They found that PNN outperformed on most of the databases in term of accuracy rate compared to other prediction techniques. Gondra [31] used Artificial Neural Network (ANN) technique to build a classification prediction model for fault-proneness. Their experiment results confirmed the performance of Support Vector Machine (SVM) over Artificial Neural Network (ANN). Wang et al. [32] built a prediction model based on Feed-Forward Neural Network (FFNN) to predict software fault-proneness of the software. Back-Propagation was used as learning algorithm for FFNN. On top of FFNN, clustering genetic algorithm (CGA) was used to extract

rules from the trained network. These rules were then used to detect fault proneness and compare predicted results to FFNN model results. The performance of FFNN models was better than the extracted rules. Khoshgoftaar [33] introduced a Feed-Forward Neural Network (FFNN) classification model to identify and isolated high-risk program.

2.5.2 Support Vector Machine

Support Vector Machine (SVM) is a data classification technique which can use small training sample to generalize high dimensional spaces. In addition to its ability to correctly classify data that is linearly separable, it is also capable of modeling non-linear relationships which are hard to be model by traditional approaches [34].

- **Elish and Elish [35]** presented a study of defect-prone prediction using Support Vector Machine (SVM). The study analyses the classification models to check whether a model is defective or not. The SVM model capability was evaluated and its performance was compared against other models, Logistic Regression (LR), K-Nearest Neighbor (KNN), Multi-Layer Perceptron (MLP), Radial Basis Function (RBF), Bayesian Belief Network (BBN), Naïve Bayes (NB), Random Forests (RF), and finally Decision Tree (DT). Four measures were used to evaluate the performance of each prediction models:

- Accuracy: the correct classification rate.
- Precision: the number of models correctly predicted to the total number of models predicted as defective.
- Recall: the number of models correctly predicted to the total number of models that are actually defective.
- F-measure: the mean of precision and recall.

The experiment was conducted on four NASA datasets and the results indicated that the models of SVM are generally better than the other models.

- **Yan et al. [36]** proposed a prediction model using Fuzzy Support Vector Regression (FSVR) to predict software defect numbers. They mentioned in their research that different empirical studies show that the defects are not distributed uniformly and the software metrics datasets are unbalanced. As a result, FSVR is developed to reduce the effect of outliers and noises of data fitting using unbalanced dataset. FSVR is a combination of regression and fuzzy logic model using software metrics to predict software quality. Fuzzy member function selection must meet the flowing constraints:
  - A membership function must be bounded in [0, 1].
  - Dataset element cannot map to different degrees of membership for one fuzzy function.

  The fuzzy membership function equation is as follows:

$$s_i = (y_i - y_{min}) * \left(\frac{(1-\sigma)}{y_i - y_{min}}\right) + \sigma\sigma = 0.01 \qquad (5)$$

- y_{min} is the minimum value of the target value set.
- σ ensures that s_i will not be zero, $s_i \in (\sigma, 1 - \sigma)$.
- y_i is large = s_i will be large.

Therefore, the more the defects there are in a module, the more the sample contributes to the regression problem. Radial Basis Function (RBF) was selected as a kernel function as it generates lower mean absolute error than linear kernel.

- **Singh et al. [37]** proposed a prediction model using Support Vector Machine (SVM) to investigate the relationship between C&K (Chidamber and Kemerer)metrics and fault proneness. Radial Basis Function (RBF) is used as a kernel function for SVM to map non-linearly data into a higher dimensional space. Five factors were used to evaluate the performance of the SVM model, sensitivity, specificity, precision, completeness, and Area Under the Curve (AUC). Their experiment results show that Coupling Between Object (CBO), Response For a Class (RFC), and Physical Source Lines of Code (SLOC) metrics have a relationship with fault proneness, while Number of Children (NOC) and Depth of Inheritance Tree (DIT) metrics were not significantly related to fault proneness.

- Xing et al. [38] proposed a classification model using Support Vector Machine (SVM) to predict fault proneness in early stage when small amount of data is available using complexity metrics. Gondra [31] performed a comparison between Artificial Neural Network (ANN) and Support Vector Machine (SVM) classification prediction models

and showed that SVM model outperforms ANN model. Twala [39] predicted software fault using five machine learning classification algorithms. Support Vector Machine (SVM) was one of these algorithms. Support Vector Machine (SVM) was one of these algorithms and performed well in their experiments. Kamei et al. [40] evaluated the performance of Support Vector Machine (SVM) prediction model and compared it against other models including logistic regression, classification tree, neural network, and linear discriminate analysis and have shown that the performance of SVM model is the best among all the tested models.

2.5.3 Genetic Algorithms

Genetic Algorithm (GA) is an optimization technique which is based on Darwinian Theory of evolution introduced by John Holland [41]. In GA, the chromosomes are competing to survive and the fittest get high chance to remain and produce progeny.

- **Bouktif et al. [7]** proposed two genetic algorithm approaches that combine/adapt a set of existing decision tree models to predict software stability.

  Genetic Algorithm for Combining Rule Set

  The first algorithm is designed to combine a set of decision tree models into a final classifier. GA chromosomes were encoded as a vector which represents the decision tree. Then crossover operation selects a random gene from one parent and adds it to the second parent. Completeness of the offspring is ensured by keeping all genes of

one of the parents and consistency is ensured by making the new genes laid over the other genes. Mutation is applied randomly to change the gene label (stable or unstable) based on defined probability and J-index was used as the fitness function.

Genetic Algorithm for Adapting a Rule Set

The second algorithm is proposed to evolve the existing decision tree models. First it selects one rule set as the initial population. The chromosome is encoded as a rule set and each condition is represented as a gene. Roulette-wheel technique is used for crossover operation as two chromosomes are selected and a random cut point is generated for each one to generate the offspring. Mutation operation is performed based on a certain probability before throwing the chromosomes into the next generation. The algorithm evaluates the new rule set by computing its J-index to generate the final model.

- **Azar and Korkmaz [8]** proposed a hybrid heuristic approach which combines three heuristic techniques namely genetic algorithm, simulated annealing, and tabu search. It uses rule-based models to predict class stability. Genetic algorithm (GA) encodes a rule set into chromosomes and represents it as a tri-partite graph. Different genetic operations are performed to generate the progeny. First it starts with elitism that copies the best chromosomes, then two crossover operations are performed to swap condition between rules, and finally the mutation operations are applied to change the node label or value. A post-process is performed on the rule set to eliminate redundancy and inconsistency. Simulated annealing (SA) approach implements

different perturbation functions on the rule set that includes removal, insertion, and replacement functions of rules and conditions. Similarly, tabu search (TS) applies neighborhood functions on the rule set which includes deletion, addition, and modification functions of rules and conditions.

- Zhong et al. [27] used Genetic Algorithm (GA) as a learning algorithm to obtain the optimization value from the random point sets and used it as initial weight for the constructed Wavelet Neural Network (WNN). Wang et al. [32] used Genetic Algorithm (GA) on the top of Feed-Forward Neural Network (FFNN) model to extract rules from the trained network. These rules were used to detect fault proneness and compare the predicting results to FFNN model's results.

2.5.4 Bayesian Network

Bayesian Network (BN), also known as Bayesian Belief Net (BBN), is a technique that represents relationship between BN variables. It is a directed acyclic graph, where each node represents an uncertain variable, and an edge represents the direct relationship between the variables [42].

- **Wagner [42]** proposed a Bayesian Network (BN) approach to assess and predict the software quality. He used Activity-Based Quality Model (ABQM) was used to break down the quality factors into detail facts and studied their influence on activities performed within thesystem. ABQM information was used to build the BN, where activity, fact, and indicator from ABQM system represent Bayesian Network nodes.

Radlinski [43] proposed a prediction model to integrate software quality features (factors) including their relationship using Bayesian Network (BN). The reason behind selecting BN as a prediction technique is its ability to incorporate expert knowledge with empirical data. Quality feature, sub-feature, and measure are all used to construct the BN. The quality feature represents the root node, whereas the sub-feature represents the children node, and finally the measure represents a leaf node. For instance, usability is used as a quality feature, effectiveness is the sub-feature of usability, and the measure is the percentage of task accomplished. Jia et al. [44] conducted an empirical study to compare ten classifier methods for fault prediction. Their experiment results show the usefulness of metric-based classification. Two classification techniques outperform all others in terms of predictive accuracy and one of which is Bayesian Network (BN).

2.5.5 Case-Based Reasoning (CBR)

Case-Based Reasoning (CBR) is an AI technique that is used to solve problems by utilizing previous knowledge and experience of similar situations or cases. Therefore, a new problem is predicted by finding the most similar case from the past [45].

- **Grosser et al. [5]** utilized Case-Based Rezoning (CBR) to propose an approach for software stability prediction. This approach implements a similarity-based comparison principle between classes to guess their chance of becoming unstable. In addition, a stress factor concept was defined which represents a major change in the requirements that can be approximated by the percentage of newly added methods.

- **Paikari et al. [46]** proposed a customized approach for CBR to predict software defect. Four factors were used to evaluate the performance of CBR models i.e., similarity function, number of the nearest neighbor cases, attribute weighted, and solution algorithm. These factors are also used to customize the CBR prediction models. The CBR models performance were evaluated and the results show that there is no difference in using different similarity functions. Finally the results also show that by using more nearest neighbor cases, the model performance can be improved.

2.5.6 Fuzzy Clustering

Fuzzy clustering is a technique which allows set of data to be a part of two or more clusters. It groups data points which populate in a multidimensional space into a specific number of clusters [47].

- **Kaur et al. [48]** used Fuzzy C-means Clustering (FCC) technique to predict software fault. The data was divided into two clusters depending on whether the data is fault prone or fault free. Grouping of the data is done based on the minimum sum of squares of distance between data and its cluster. The performance of FCC model is better than the performance of k-means model. Yuan and Ganesan [49] predicted number of faults in a software using Fuzzy Subtractive Clustering (FSC) technique and module-order method to classify modules as fault-prone or fault-free. Fuzzy rules were generated using data points to predict the number of faults. Sugeno method was

used as fuzzy system interface that involves the following processes: fuzzify inputs, apply fuzzy operator, apply implication method, and defuzzify.

2.5.7 Classification-Regression Trees (CART)

Classification-Regression Trees (CART) is a binary recursive technique which enables processing of continuous data as target and predictors. Starting from the root node, the data is splits into two children and each child splits into grandchildren and so on until it reaches the maximal-size. CART can generate a sequence of nested trees and the optimal tree can be evaluated based on the predictive performance [50].

- **Khoshgoftaar et al. [51]** introduced the CART algorithm and used it to predict fault-proneness of model over several releases. A case study was developed to evaluate the CART prediction models on two classification trees. The classification tree is represented as a tree of decision rules which classifies whether the model is fault-prone or not. Each node represents a decision, and each edge represents decision's result and each leaf is labeled as fault-prone or not. The decision starts from the root node and CART traverses a downward path in the tree until it reaches the leaf node and then returns the decision.

2.6 Summary

All Artificial Intelligence techniques are summarized in this section. Table 2-1 introduces the research's author names, the used AI technique (s), the objective of the research, and finally the data that was used in the research's experiments.

TABLE 2-1: Literature Review Summary Table

Authors	AI Technique	Objective	Experiment Data
Zhong et al. [27], 2011	Wavelet Neural Network (WNN)	Predict software quality	MPD database and JM1 product model
Thwin and Quah[6], 2005	Ward Neural Network (WNN), Generalized Regression Neural Network (GRNN)	Predict number of defects and the lines changed per class	HMI & QUES systems
Kanmani et al. [28], 2007	Back-Propagation Neural Network (BPNN), Probabilistic Neural Network (PNN)	Predict fault proneness	In-hose software
Peng et al. [29], 2009	Fuzzy Neural Network (FNN)	Predict the relationship between software metrics and quality factors	Sample data
Al-Jamimi and Ghouti [30], 2011	Probabilistic Neural Network (PNN)	Predict fault proneness	NASA database

Authors	AI Technique	Objective	Experiment Data
Gondra [31], 2007	Artificial Neural Network (ANN)	Predict fault proneness	NASA database
Wang [32], 2004	Feed-Forward Neural Network (FFNN)	Predict fault proneness	Telecommunications System
Khoshgoftaar [33], 1994	Feed-Forward Neural Network (FFNN)	Predict fault proneness	Telecommunications System
Elish and Elish [35], 2008	Support Vector Machine (SVM)	Predict defect prone	NASA database
Yan et al. [36], 2010	Fuzzy Support Vector Regression (FSVR)	Predict number of defect in the software	MIS and RSDIMU systems
Singh et al. [37], 2010	Support Vector Machine (SVM)	Predict fault proneness	NASA database
Xing et al. [38], 2005	Support Vector Machine (SVM)	Predict fault proneness	MIS system
Gondra [31], 2007	Support Vector Machine (SVM)	Predict fault proneness	NASA database

Authors	AI Technique	Objective	Experiment Data
Twala [39], 2011	Support Vector Machine (SVM)	Predict fault proneness	NASA database
Kamei et al. [40], 2006	Support Vector Machine (SVM)	Predict fault proneness	Company software
Bouktif et al. [7], 2004	Genetic Algorithm (GA)	Predict software stability	In-house software
Azar and Korkmaz [8], 2010	Genetic Algorithm (GA)	Predict class stability	CDK, Jedit, Jetty, Winnie, Jrate, JDK
Zhong et al. [27], 2011	Genetic Algorithm (GA)	Predict software quality	MPD database and JM1 product model
Wang et al. [32], 2004	Genetic Algorithm (GA)	Predict fault proneness	Telecommunications System
Wagner [42], 2010	Bayesian Network (BN)	Predict software quality	NASA database
Radlinski [43], 2011	Bayesian Network (BN)	Integrated software quality prediction	N/A
Jia et al. [44], 2009	Bayesian Network (BN)	Predict fault proneness	QMP
Grosser et al. [5], 2003	Case-Based Reasoning (CBR)	Predict software stability	JDK

Authors	AI Technique	Objective	Experiment Data
Paikari et al. [46], 2011	Customized Case-Based Reasoning (CBR)	Predict software defect	NASA database
Kaur et al. [48], 2010	Fuzzy C-mean Clustering (FCC)	Predict fault proneness	NASA database
Yuan and Ganesan [49], 2000	Fuzzy Subtractive Clustering (FSC)	Predict number of defect in software	Telecommunications System
Khoshgoftaar et al. [51], 1999	Classification-Regression Trees (CART)	Predict fault proneness	Telecommunications System

CHAPTER 3

BACKGROUND

Stability is one of the most important quality factors to consider in reducing maintenance cost and efforts. Object-oriented class forms the basic components of the software systems. Hence, stable classes may contribute to reducing the maintenance cost of the system. Consequently, many metrics have been proposed to measure the class stability. This chapter presents different definitions of software stability, describes class stability metrics and how class stability is measured, defines measurement validation, and finally discusses different types of stability evaluation.

3.1 What is stability?

As mentioned earlier, stability is one of the most important quality factors. Software stability has been defined in many forms by researchers. Grosser et al. [5, 52]

defined stability as the ability of a software to evolve without changing its design. Fayad [11-13] stated that a software product can be stable if the inclusion of changes does not lead to re-engineering of the product. Localized changes make programs more stable to resist the potential ripple effect of program modification, this approach is supported by Yau and Collofello [14]. Soong [15] defined stability as the resistance to changes amplification for a given program. Measuring the difficulty in changing a module is the stability definition of stability as given by Martin [53]. Elish and Rine indicated that avoiding inter class propagation changes during the design makes software stable [9]. It can be noted that software stability has been defined in many forms by researchers. Hassan [54] outlined three general definitions of software stability. The first definition is the resistance to any change made to software. The definitions of Soong [15] and Martin [53] correspond to this approach. The second definition is avoiding the ripple effects of the new modifications. Fayad [11-13] and Yau and Collofello [14] followed this approach. The last approach refers to the changes that are made to the software during its evolution and not in the development cycle. Grosser's [5, 52] and Elish and Rine [9] definitions are in line with this approach.

3.2 Class Stability Metrics

Object-oriented classes are the basic elements of software architecture. Stable classes may contribute to reducing the maintenance cost and efforts. Therefore, many software metrics were proposed by researchers to measure the stability of the object-

oriented class. This section discusses in details the existing class stability metrics and how they measure the class stability.

3.2.1 Class Implementation Instability Metric (CII)

Li et al. [3] proposed class implementation instability (CII) metric to measure the evolutionary changes of a class. It uses the Lines of Code (LOC) metric to measure the percentage of changes in line of a code between a class in version N and version N +1. To measure class stability using CII metric, two positive integers need to be considered:

- (q) The LOC of class A version N
- (r) The LOC of class A version N + 1

Once class with version N + 1 is registered, the CII metric can be measured as following:

$$[(r - q) / q * 100] \qquad (6)$$

CII metric output can be represented as a real number of LOC percentage between version N and N +1.

3.2.2 Class Stability Metric (Grosser)

Grosser et al. [5] proposed an approach to predict stability using case-based reasoning (CBR). As part of CBR, they designed a class stability metric that is based on class interfaces or public and protect methods. So, the class is considered stable class if its interface remains unchanged between versions. To measure class stability using Grosser metric, assume c is a class and $I(c_i)$ is the interface of c in version i. The class

stability can be calculated by comparing I(c_i) to I(c_{i+1}) the following version, as represented by the following equation:

$$\boldsymbol{NS(c_{i\rightarrow i+1}) = \frac{\#\cap(c_i, c_{i+1})}{\#\, I(c_i)}} \qquad \textbf{(7)}$$

$$\textit{Where} \cap\,(\boldsymbol{c_i, c_{i+1}}) = \boldsymbol{I(c_i) \cap I(c_{i+1})}$$

NS is the stability output that ranges between 0 (absolute unstable) and 1 (absolute stable).

3.2.3 Class Stability Metric (Stab)

Daniel et al. [4] proposed a metric that is based on number of method (NOM) to measure the stability of the class. According to them, stability is affected if one method is added or removed from the class when comparing to the pervious class version. The following equation is used, where M is the NOM and C is class:

$$(\boldsymbol{i} > 1)\, \boldsymbol{Stabi(C, M)} = \begin{cases} \mathbf{1}, & \boldsymbol{Mi(C) - Mi - 1(C) = 0} \\ \mathbf{0}, & \boldsymbol{Mi(C) - Mi - 1(C) \neq 0} \end{cases} \textbf{(8)}$$

The class is considered stable if the metric output is 1, otherwise it is unstable if it is 0. Furthermore, they used the class history (a fraction of the number of class versions) as an overall indicator of stability. The following equation is used, where n is total number of versions:

$$(\boldsymbol{n} > 2)\boldsymbol{Stab1 \ldots n(C, M)} = \frac{\sum_{i=2}^{n} \boldsymbol{Stabi(C,M)}}{\boldsymbol{n-1}} \qquad \textbf{(9)}$$

3.2.4 Class Stability Metric (CSM)

Alshayeb et al. [2] proposed class stability metric (CSM) based on class properties that affect the stability. Eight class factors were identified to measure the stability of the class. Table 3-1 illustrates these properties.

TABLE 3-1: Class Stability Metric (CSM) properties

CSM	
Class access-level	Class variable Access-level
Class interface name	Method Signature
Inherited class name	Method Access-level
Class variable	Method body

Different types of changes can occur in the class: addition, deletion, modification, and unchanged. CSM uses the unchanged type to count unchanged properties between class versions to measure the stability. Table 3-2 shows how CSM counts the unchanged class properties.

TABLE 3-2: Final Class Properties and Classification

Metric	How to classify		Count
Class Access-Level	Unchanged	if the class access-level has not been modified or deleted from version *i* to version *i+1*.	+1
Class interface name	Unchanged	if the interface name has not been modified or deleted from version *i* to version *i+1*.	+1
Inherited Class Name	Unchanged	if the inherited class name has not been modified or deleted from version *i* to version *i+1*.	+1
Class variable	Unchanged	if the class variable has not been modified or deleted from version *i* to version *i+1*.	+1
Class Variable Access -Level	Unchanged	if the class variable access-level has not been modified or deleted from version *i* to version *i+1*.	+ 1
Method Signature	Unchanged	if the method signature has not been modified or deleted from version *i* to version *i+1*.	+ 1
Method Access-level	Unchanged	if the method access-level has not been modified or deleted from version *i* to version *i+1*.	+1
Method Body (Code)	Unchanged	if number of lines and the contents for a method did not change from version *i* to version *i+1*.	+1

To measure class stability, CSM calculates the maximum-possible-changes that can happen to each property with respect to class version i, Table 3-3 lists the maximum-possible-changes and their values. Each property in version i+1 will be compared to the same property in version i and divided by the maximum-possible-changes to the same property in version i to compute the extent-of-stability as following:

$$Stab_{Property} = Unchanged_{Property} / Number_{Property} \quad (10)$$

Where**,**

- **StabProperty** is stability of class property
- **UnchangedProperty** is the number of unchanged items of that property
- **NumberProperty** is the maximum-possible-changes value

TABLE 3-3: Maximum Possible Changes for Each Property in version i

Property	**WHEN maximum possible changes happen**	**The maximum possible change value**
Class Access-level	Class access-level has been modified from the version *i* to version *i+1*.	1
Class interface name	All interfaces have been modified or deleted from version *i* to version *i+1*.	Total number of interfaces in version *i*.
Inherited Class name	All inherited class names have been modified or deleted from version *i* to version *i+1*.	Total number of inherited class names in version *i*. (In JAVA there is no multiple inheritance So always, the value is one)
Class variable	All the variables have been modified or deleted from version *i*`to version *i+1*.	Total number of variables in version *i*.
Class variable Access-level	All the variables are present in version *i* but their access-levels have been modified from version *i* to version *i+1*.	Total number of class variables in version *i*.

Property	WHEN maximum possible changes happen	The maximum possible change value
Method Signature	All methods signatures have been modified or deleted from version i to version i+1.	Total number of methods in version i.
Method Access-level	All methods are present in version i, but their access-levels have been modified from version i to version i+1.	Total number of methods in version i.
Method Body (Code)	All methods are present in version i, but at least one line of code in each method has been deleted or modified from version i to versioni+1.	Total number of methods in version i.

Once the stability of each property is defined, CSM sums all values and divide by eight (the total number of the identified properties) to find the overall class stability metric as follows:

$$Stability_{CLASS} = \frac{Stab_{ClassAL} + Stab_{Interface} + Stab_{Inhr} + Stab_{Mthd} + Stab_{Var} + Stab_{varAL} + Stab_{MthdAL} + Stab_{Body}}{properties_{CLASS}}$$

(11)

Where,

- **StabClassAL** is class access-level stability
- **StabInterface** is class interface stability
- **StabInhr** is inherited class name stability
- **StabMthd** is method signature stability
- **StabVar**is class variable stability
- **StabVarAL** is class variable access-level stability
- **StabMthdAL** is method access-level stability
- **StabBody** is method body stability
- **PropertiesClass** = 8 the identified class properties.

The class is considered completely stable if CSM output is 1, and unstable if the output is 0.Moreover, CSM can be used to measure the class instability by using the complement of class stability (1 – class stability).

3.3 Stability Evaluation

Stability is one of the most important quality factors of any software system. Evaluating software stability helps in understanding the system evolution and identifying future threats. Class attributes such as cohesion, coupling, etc. can be used to evaluate the stability factor by measuring the changes impact on software system. Two approaches exist for stability evaluation in literature, retrospective and predictive [55, 56]. This section illustrates both approaches.

3.3.1 Retrospective

Retrospective evaluation looks at events that have already occurred and have taken place. In software engineering, retrospective can be used to compare class properties from one version to the following version and evaluate different quality factors such as stability. It helps to determine and improve the quality of the software. It can be used to perform empirical validation and can be the basic source of data to predict the evaluation. Measuring the stability of the class between different versions using class stability metrics is an example of retrospective evaluation.

3.3.2 Predictive

Predictive evaluation provides a precise assessment of the software by evaluating expected changes and how the software can adapt to these changes. It is considered as a preventive process as it attempts to understand potential threats. Software metrics are the key to building a prediction model. Different types of metrics can be used such as complexity, coupling, and cohesion metrics. In software quality, the aim of predictive evaluation is to predict the software quality as close to actual product quality as possible. Consequently, artificial intelligence techniques have been used by many researchers to build a prediction model to assess the software quality.

3.4 Measurement Validation

Validation assures that measurements assess what they are supposed to measure. Two types of validation i.e., empirical and theoretical that are used to validate the measurement.

- **Theoretical validation:** The measure is said to be theoretically validated if it respects a certain defined criteria as outlined by Kitchenham and Lawrence [23]. They proposed a framework for software measurement validation using four criteria:

1. *"For an attribute to be measurable, it must allow different entities to be distinguished from one another",* there must exist two entities for which the measure results in different values.
2. *"A valid measure must obey the Representation Condition", "*preserves the concept of the attribute and the way in which it distinguishes different entities".
3. *"Each unit of an attribute contributing to a valid measure is equivalent".*
4. *"Different entities can have the same attribute value",* two classes with the same entities will have the same stability value.

- **Empirical validation:** An empirical validation usually includes experimentation and hypothesis testing and can be defined as *"Which corroborates that measured values of attributes are consistent with values predicted by models involving the attribute"*[23]. Empirical validation is the process of establishing the accuracy of a prediction system by comparing the model performance with known data in a given environment [21, 22].

Building a prediction model using software metrics can be used to obtain assurance about software quality [6]. The objective of this research is to predict the object-oriented class stability using software metrics as measured by the class stability metrics. To achieve this objective, a research methodology is developed to design the study as follows:

- **Proposed research hypotheses:** A hypothesis needs to be developed to provide an articulation of a theoretical basis which relates the software metrics to stability [57].
- **Data collection and analysis:** The experimental data has an important role to evaluate the model accuracy. Therefore, data will be collected from different resources and number of software metrics will be extracted based on previous research studies. The extracted software metrics may have sort of correlation, "If a group of variables in a data set are strongly correlated, these variables are likely to measure the same underlying dimension (i.e., class property)" [57]. Hence, correlation between extracted metrics needs to be identified in order to reduce the number of software metrics.
- **Experiment setup:** To select the techniques that will be used to build the prediction models and specify the setup parameters. The experiment dataset will be created using the selected software metrics as predictor variables and stability measurement output as target variable.
- **Experiment Results and observations:** Once the experiment is completed, the results will be analyzed and the accuracy of the models will be evaluated using different methods such as cross-validation or hold-out sample of dataset [23].

3.5 DTREG & Minitab

DTREG and Minitab tools are selected to build the prediction models for the research experiment. DTREG is a classification and regression tool that is used to describe data relationships and can be used to predict values for future observations [58]. Figure 3-1 shows DTREG main screen.

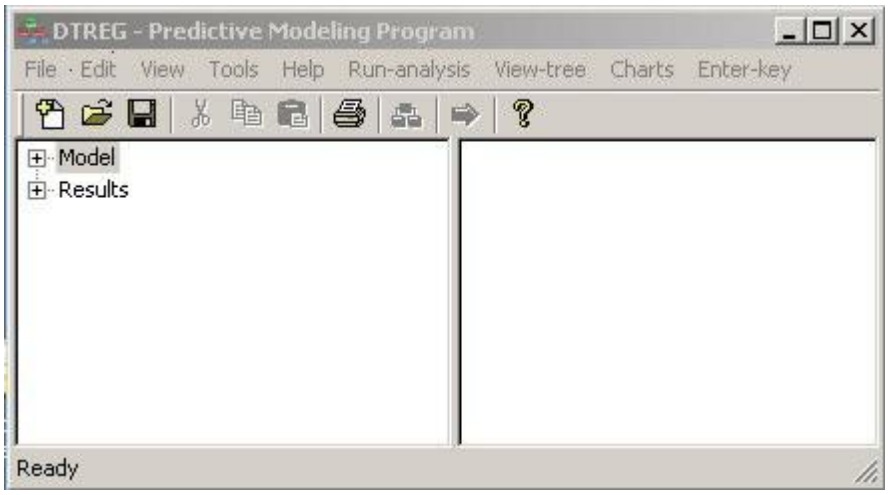


Figure 3-1: DTREG main screen

DTREG can analyze data and generate a model which helps to predict the values of the target variable using the values of the predictor variables. DTREG provides different modeling methods such as neural networks, support vector machines (SVM), decision trees, gene expression programming, and logistic regression. Two artificial intelligence techniques are selected: Multilayer Perceptron neural network (MLP) and Support Vector Machine (SVM) to build prediction models for the research experiment. Minitab is selected to perform Linear Regression (LR) analysis. Linear regression is the

most widely used predictive model. For this reason, an experiment is conducted on stability using multiple linear regressions to observe the result and compare it to the AI techniques. This section descries the selected techniques.

3.5.1 Multilayer Perceptron Neural Network (MLP)

Multilayer Perceptron (MLP) neural network is a feed forward model that maps sets of input data into a set of output values through a hidden layer. It is a fully-connected network that consists of multiple layers of nodes, input, hidden, and output layers, in a directed graph. The values move in one direction, from input to hidden to output layers and no values are fed back to earlier layers. Figure 3-2 illustrates MLP architecture [58].

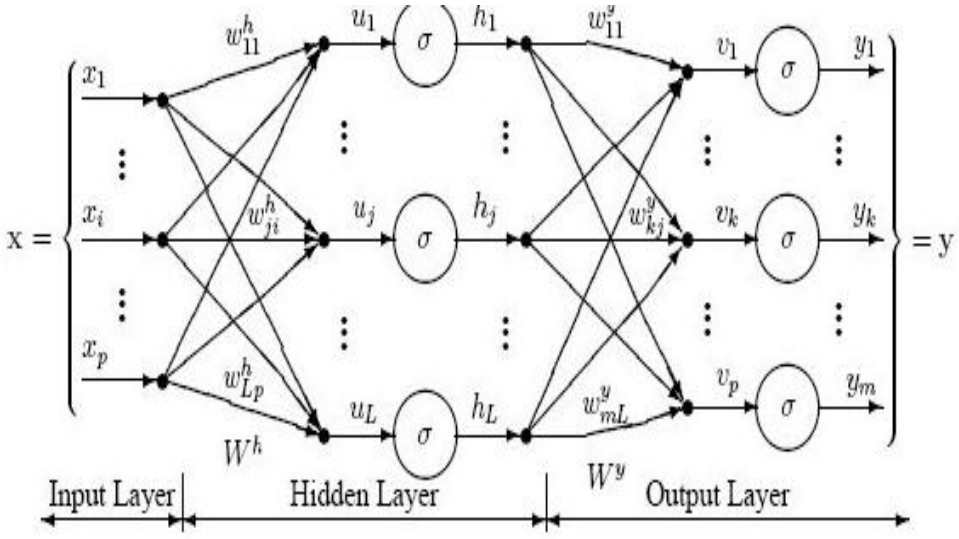


Figure 3-2: Multilayer Perceptron architecture

MLP is one of the most widely used types of neural networks and consists of input, output, and hidden layers. Hidden layer is the most important part of MLP and the number of the hidden layers may vary. Defining the number of neurons in the hidden layer is a challenging task because if large number of neurons is used, then over-fitting will occur and the model will memorize instead of predicting during the training process, whereas selecting inadequate number of neurons will result in poor fitting model. DTREG provides an option to find the optimal number of neurons in the hidden layer automatically. To be able to find the number of neurons automatically, DTREG requires a few parameters to be set by the user, minimum and maximum number of neurons in the hidden layer, number of neurons to be added in each trial, and number of neurons for which the trial should stop if there is no enhancement in the model. DTREG uses Differential Evolution method which is based on genetic algorithm technique to find the optimal starting weight values. It first starts by creating an initial population with random weights pairs of weight sets are then combined to produce new weight values. A random mutation process will then be applied to the generated weights. Next, each weight is evaluated to see how well it optimizes the function. The best weight found will have a high probability to be selected for the next generation. This process is repeated until a set of weights is found that sufficiently minimizes the error. Once the optimal weight values are found, then convergence algorithms that are provided by DTREG are used to find the optimal solution.

3.5.2 Support Vector Machine (SVM)

Support vector machine (SVM) is an AI technique that is used for data analysis, pattern recognition, classification, and regression analysis. SVM builds a model to represent the input data by assigning it into one category or the other. Figure 3-3 illustrates SVM process.

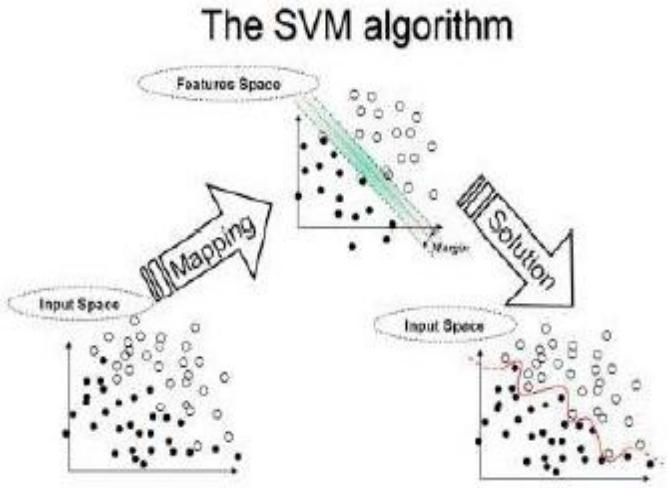


Figure 3-3: Support Vector Machine process

SVM constructs an N-dimensional hyper-plane that optimally separates the data into two categories. The kernel function such as linear, Radial Basis Function (RBF), and sigmoid is used to transform input data into an n-dimensional space. RBF and Sigmoid kernel functions can handle non-linear relationship between the target variable and the predictor variables as they have the same behavior for certain parameters. The model parameters have positive effect on the accuracy of an SVM model. Therefore, DTREG provides two methods, grid search and a pattern search to find optimal parameter values. A grid search utilizes geometric steps to use the different values of each parameter across

the specified search range, whereas a pattern search focuses on the center of the search range to make trial steps in each direction for each parameter. Both methods are selected to be part of the SVM experiment. Grid search will first be used to find a region near the global optimum point, and then pattern search is performed by utilizing the best point found by the grid search to find the global optimum by starting in the right region.

3.5.3 Multi Linear Regression (LR)

Linear regression is the most widely used predictive model. It is designed to minimize the sum of the squared errors to fit a straight line to a set of data points. The form of the function is:

$$Y = B_0 + B_1 * X_1 + B_2 * X_2 + \ldots + B_n * X_n \qquad (12)$$

Y is the target or output variable, where X_1, X_2,… X_n are the predictor variables such as software metrics, and B_0 is a constant. The goal of linear regression is to minimize the sum of the squared error which is estimated by the difference between the actual value of the target variable and its predicted value. Linear regression models are simple to implement and training the model is usually much faster than other regression methods such as neural networks.

3.5.4 Logistic Regression (LogR)

Logistic regression is also one of the most widely used classification model. It is a type of regression analysis used to predict the outcome of a categorical variable based on one or more predictor variables. Logistic regression has also been used to classify observations into one or two categories, and it may give fewer classification errors than discriminate analysis in some cases.

CHAPTER 4

DATA COLLECTION & ANALYSIS

There are many programming languages which are used to build Object-Oriented software, Java is one of the most active language currently used. Classes are form the basic components of software and keeping the class stable contribute towards reducing the maintenance cost and effort. Therefore, out of the different software systems available, Java classes have been chosen to build the prediction model in order to predict the object-oriented class stability using software metrics**.** This chapter introduces different software tools that are used to extract metrics form Java classes and also discusses the different phases of data collection and analysis.

4.1 Software Metric Tool

Software metrics can be used to provide valuable feedback on the quality of the source code. They also enable the software team to assess the quality of their project. Therefore, many software metric tools have been developed to analyze source code, collect software metrics, and assist in measuring software quality. This section describes different tools that are used in this research to collect the metrics and measure the correlation between these metrics.

4.1.1 Loc Metrics

LocMetrics is a software tool that is designed to calculates the total lines of code and can also find out the number of blank lines, comment lines, both code and comment lines, logical lines of code, comment words, and physical executable lines of code which is calculated by subtracting the number of blanks and comment lines from the total lines of source code. This tool is used to extract the physical executable lines (LOC) metric from the source codes. Figure 4-1 shows the main screen.

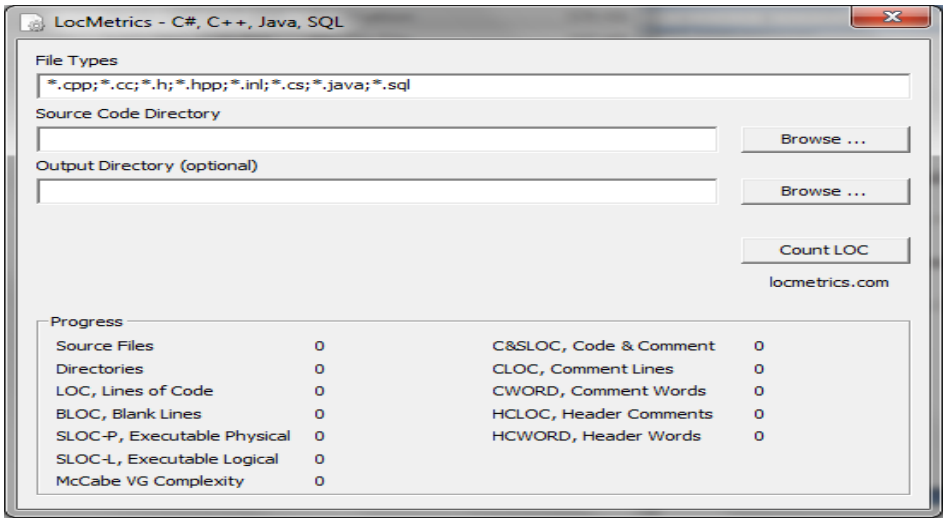


Figure 4-1: LocMetrics main screen

4.1.2 LOCC

LOCC is a software metric tool that first analyzes the source code and then calculates the number of attributes such as packages, classes, methods in each class, and the total number of lines. This tool is used to calculate the number of methods of each class and to find the internal classes. Figure 4-2 shows the main screen.

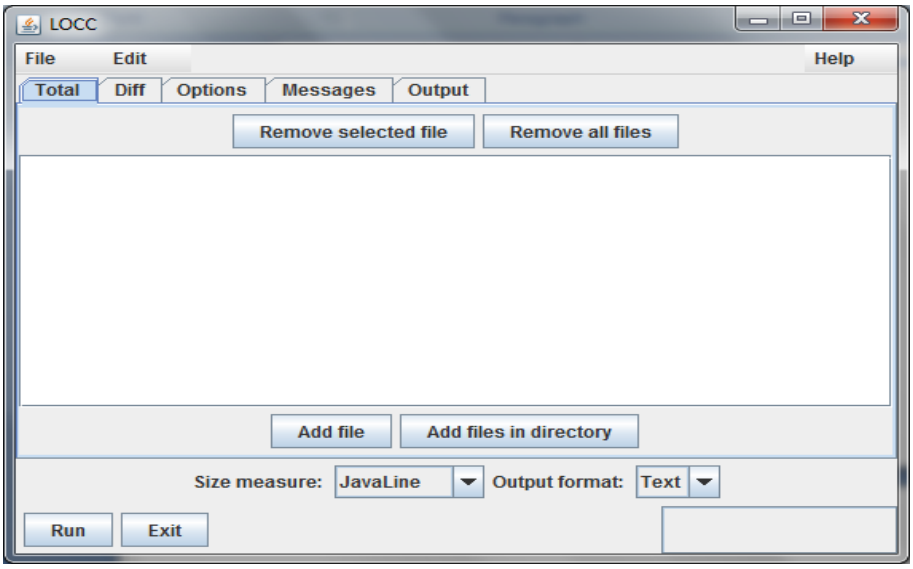


Figure 4-2: LOCC main screen

4.1.3 Metamata

Metamata is a tool designed for Java environment. It has many features, but the most important for our research is the metric feature which calculates many software metrics from Java source code. Hence, this tool is used to extract software metrics from Java classes in all projects. Figure 4-3 shows the main screen of Metamata tool.

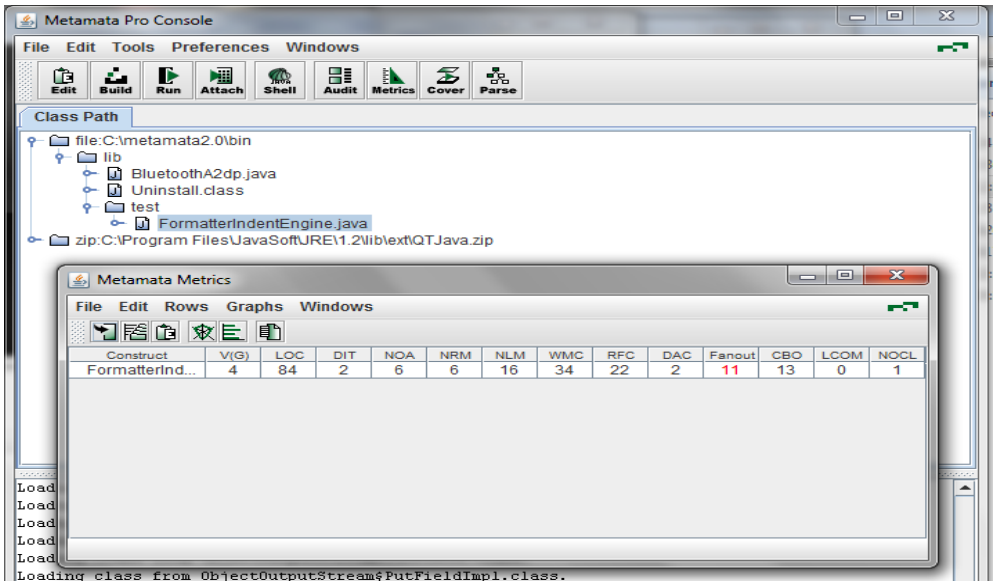


Figure 4-3: Metamata main screen

4.1.4 OOMeter

OOMeter is a software metric tool has been proposed by Alghamdi [59]. It extracts and measures class attributes for Java code. It supports different types of object-oriented software metrics such as cohesion, coupling, and size complexity. This tool is used to parse and analyze different versions of Java source code and prepare the required data for the CSM tool which is then used to measure the class stability. Figure 4-4 shows the architecture of OOMeter.

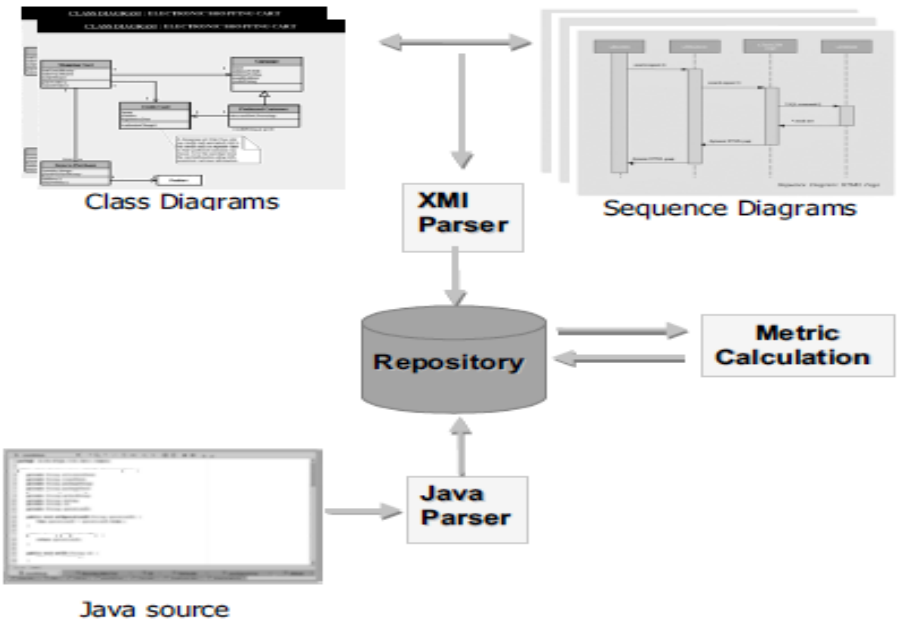


Figure 4-4: OOMeter architecture

4.1.5 CSMT

CSMT is a Java-based tool used to measure the class stability [60]. In addition to OOMeter data, CSMT uses LOCC "Lines of Code Counter" tool to calculate the number of lines of codes per method. The collected data is used later to find the method signature and the method body stability in order to measure the class stability. Figure 4-5 shows CSMT Framework.

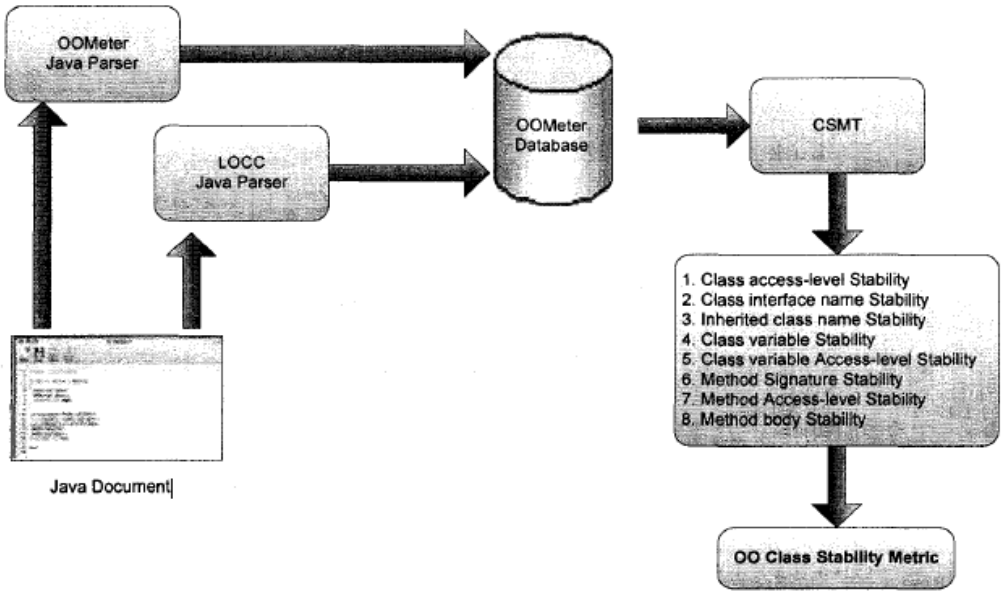


Figure 4-5: CSMT Framework

4.2 Data Collection

To predict the object-oriented class stability, three open source Java projects (Android, Eclipse, and NetBeans) with four different versions for each project are selected to create the required dataset for the research experiment. 1336 Java classes are selected from these projects. The same classes are collected from each version. The dataset consists of stable and unstable classes. The file size of the class was used as an indicator to differentiate between stable and unstable class. For instance, if the file size of the class changed between its versions, that means there are changes on this class which may affect its stability. Most of the classes in the Android project are stable where as Eclipse and NetBeans projects have a combination of both stable and unstable classes. Table 4-1 describes the selected Java projects.

TABLE 4-1: Selected Java projects

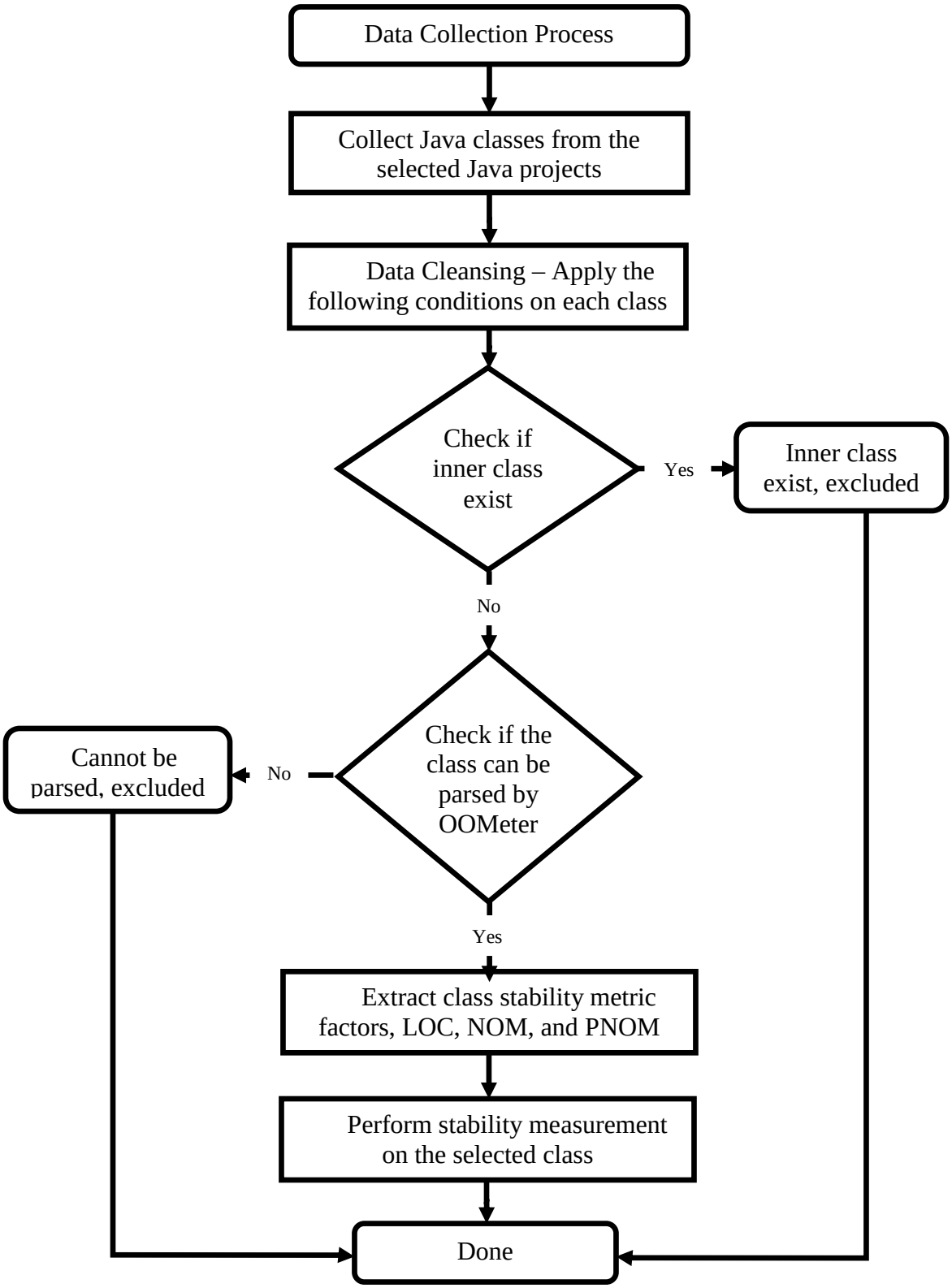
Java Projects	Version	Number of Class
Android	1.5 – 1.6 – 2.1 – 2.3.1	532
Eclipse	2.0 – 2.0.2 – 3.5 – 3.6	368
NetBeans	3.5.1 – 4.0 – 5.0 – 5.5.1	436

Android is a software stack for mobile devices that aims to develop low cost technologies and user friendly applications. It enables developers to work collaboratively to produce qualified products that meet customer's need at a much lower cost [61].

Eclipse is a development tool (IDE) for Java that was created by an open source community to support open source products and services. It has the ability to extend and add new functionalities and can also be used to create general purpose applications [62].

NetBeans is an IDE which is dedicated to developing software products that address the needs of developers, users and the businesses. It supports different languages and frameworks [63].

This section discusses the different phases of data collection process. The following flowchart illustrates the data collection phases.


Data Collection Process
Collect Java classes from the selected Java projects
Data Cleansing – Apply the following conditions on each class
Check if inner class exist
Yes
Inner class exist, excluded
No
Check if the class can be parsed by OOMeter
No
Cannot be parsed, excluded
Yes
Extract class stability metric factors, LOC, NOM, and PNOM
Perform stability measurement on the selected class
Done

4.2.1 Collect Java Classes

Java classes with different versions are collected from different applications in each Java project to build a dataset for the research experiment. The classes extracted from the Java applications of android project are described in Table 4-2.

TABLE 4-2: Android applications

Android project	
Main Application	**Application**
Android	app – appwidget – bluetooth – content – database – graphics hardware - location – os – view - widget content → res database →sqlite view →animation view →inputmethod
Com →android	browser – calendar – camera – email – phone email → activity → setup
Java	lang – security – text
Javax→ xml	parsers

Also, many classes with different versions are collected from the Eclipse applications as shown in Table4-3.

TABLE 4-3: Eclipse applications

Eclipse project	
Main Application	**Application**
org.eclipse.compare	compare – internal – structuremergeviewer
org.eclipse.debug.core	debug
org.eclipse.help	context – toc – util
org.eclipse.pde.runtime	registry – runtime
org.eclipse.swt→swt	graphics – wedgets
org.eclipse.jdt.core	antadapter – batch – formatter
org.eclipse.jdt.core→compiler	ast – classfmt – flow – impl - parser

Finally, classes are also collected from the NetBeans applications as shown in Table4-4.

TABLE 4-4: NetBeans applications

NetBeans project	
Main Application	**Application**
NetBeans	ant – autoupdate – beans – classfile – action – form – actions filesystems – pertftool – java – monitor – properties
NetBeans→ form	Actions – edotor – layoutsupport - palette layoutsupport→delegates
NetBeans→utilities	search - type

4.2.2 Data Cleansing

To build a dataset that can be used by prediction models, the following points need to be taken into account:

- The selected java class should not have an inner class, as this may affect the measurement accuracy of some class stability metrics.
- The selected java class can be parsed by the OOMeter tool. OOMeter is used to extract class structural attributes such as local variables and method signatures.

Data cleansing phase is performed manually and is designed to ensure that all the collected classes meet the above mentioned points. Each class goes through two steps before it is selected. First step is checking for the presence of an internal class by using LOCC tool. If an internal class presence is found then that class is excluded, as some class stability metrics are not supported with codes that have internal classes. Second step is to parse the class using the OOMeter tool, if an error is received from the parsing process, then that class is excluded as CSM metric cannot measure class stability if the class cannot be parsed by an OOMeter tool. As a result, when the class is excluded other versions of the same class will be excluded as well.

4.2.3 Class Stability Metric Factors

Class stability metrics are designed based on class factors. This phase is developed to extract these factors from the collected Java classes. CSM metric is excluded from this phase as the required factors will be automatically extracted by OOMeter and CSMT tools during the stability measurement phase. LocMetrics tool is used to extract lines of code (LOC) to measure the class stability using CII metric. Executable lines of code are taken into account and other lines such as comments and blank are excluded. LOCC tool is used to extract the number of methods (NOM) to measure class stability using Stab metric. Finally, public and protected methods are manually extracted from each class as the Stab metric measures the stability of the class based on adding or removing method between class versions. Table 4-5 shows an

example of extracted factors of the class stability metrics from different class versions. Appendix A contains the details for the selected Java projects.

TABLE 4-5: Sample of class stability factors

Class Name	Class Attributes											
	2.0			2.0.2			3.5			3.6		
	P-NOM	LOC	NOM	P-NOM	LOC	NOM	P-NOM	LOC	NOM	P-NOM	LOC	NOM
BufferedContent	9	53	9	9	53	9	9	53	9	9	53	9
CompareConfiguration	26	174	27	27	174	27	33	341	35	33	341	35
CompareUI	12	66	13	12	66	13	22	113	25	22	113	25
CompareViewerPane	5	57	6	5	57	6	20	170	21	20	170	21
HistoryItem	6	30	6	6	30	6	8	47	8	8	47	8
NavigationAction	4	27	4	4	27	4	4	47	4	4	47	4
ResourceNode	14	92	14	14	92	14	17	119	17	17	119	17

4.2.4 Stability Measurement

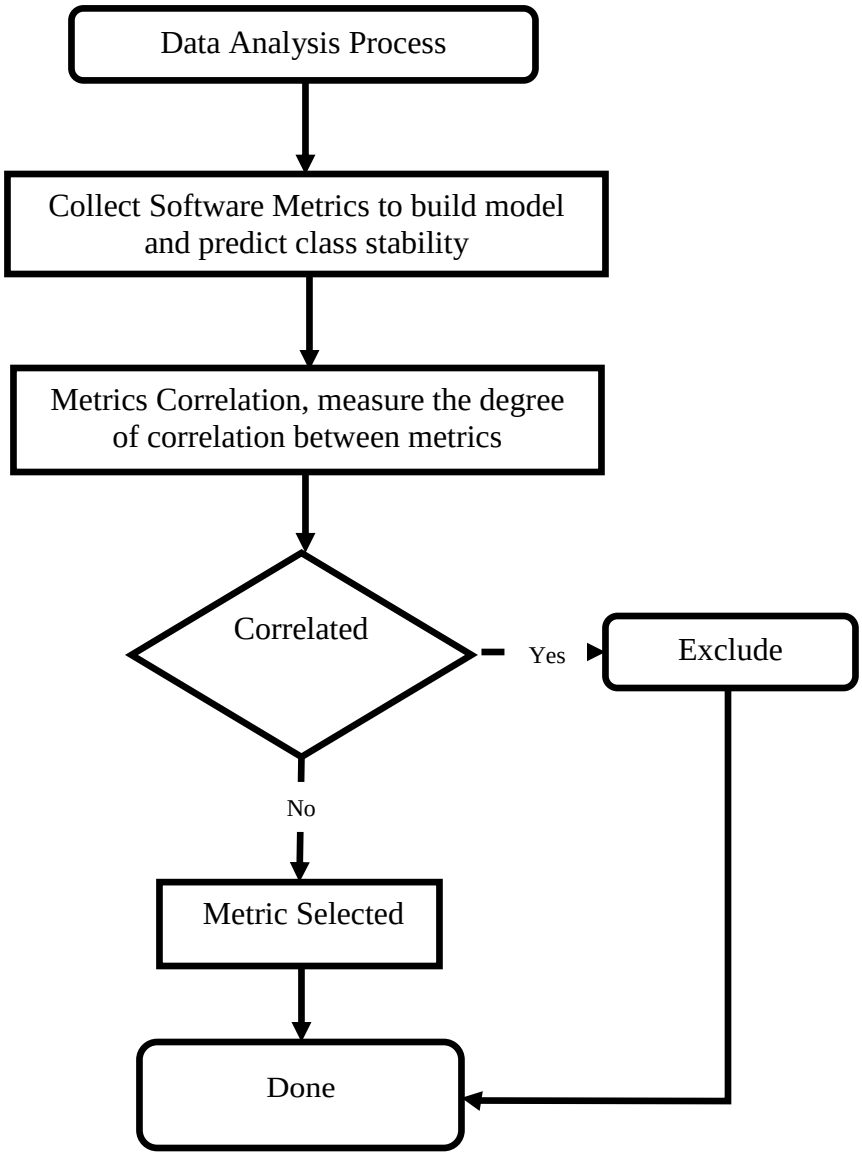
After the stability metric phase is completed, the stability measurement phase starts. A retrospective analysis is done by measuring the class stability for all collected classes. The measurement process is based on various comparable properties of the classes through various versions. It is performed manually on collected classes to produce the output values of CII, Grosser, and Stab metrics. The CSM metric uses OOMeter and CSMT tools to extract the required class factors then the class stability is measured. Table 4-6 shows the sample of the stability measurement results. Stability results for the selected Java projects are in Appendix B.

TABLE 4-6: Sample of stability measurement results

Class Stability Measurement												
Class Name	CSM			CII (LOC)			Stab (NOM)			Grosser(P-NOM)		
	3.5.1/4.0	3.5.1/5.0	3.5.1/5.5.1	3.5.1/4.0	4.0/5.0	5.0/5.5.1	3.5.1/4.0	4.0/5.0	5.0/5.5.1	3.5.1/4.0	4.0/5.0	5.0/5.5.1
IDESettingsBeanInfo	0.9375	0.78125	0.729167	-16.6667	14.28571	7.5	1	1	1	1	1	1
NbPlaces	0.988095	0.667857	0.655612	17.94872	-29.7101	0	1	0	1	1	0.58	1
NonGuiMain	0.992188	0.800781	0.742188	0.274725	0.273973	0	1	1	1	1	1	1
Plain	0.796875	0.739583	0.710938	-4.61538	0	0	1	1	1	1	1	1
WarmUpSupport	0.9375	0.78125	0.729167	50	0	0	1	1	1	1	1	1
AboutAction	0.65625	0.46875	0.4375	35	11.11111	276.6667	0	1	0	1	0.8	1
ConfigureShortcutsAction	1	0.796875	0.739583	19.35484	-37.8378	0	0	1	1	1	1	1
GlobalPropertiesAction	0.525	0.3875	0.341667	-7.14286	0	0	1	1	1	0.8	1	1
HTMLViewAction	0.6875	0.53125	0.479167	6.849315	0	0	0	1	1	1	1	1
SystemExit	0.775	0.6375	0.591667	0	16.66667	0	1	0	1	0.8	1	1
ViewRuntimeTabAction	0.825	0.725	0.691667	3.846154	-3.7037	0	0	1	1	0.6	1	1

4.3 Data Analysis

The goal of this phase is to select the software metrics that has a relationship with stability as a quality factor. These metrics are used later to build the AI prediction models and to predict the stability as measured by object-oriented class stability metrics. This section discusses the data analysis process. The following flowchart illustrates data analysis phases:


Data Analysis Process
Collect Software Metrics to build model and predict class stability
Metrics Correlation, measure the degree of correlation between metrics
Correlated
Yes
Exclude
No
Metric Selected
Done

4.3.1 Collect Software Metrics

Many software metrics have been proposed and used by practitioners in several research areas such as software measurement. Software metric assign numbers to class attributes such as number of methods (NOM), lines of code (LOC), lack of cohesion methods (LCOM), etc. In this phase many software metrics are selected which belong to different metric categories such as coupling, cohesion, inheritance, and complexity. An example of these metrics is the C&K suite of metrics that was proposed by Chidamber and Kemerer [64]. In addition to class stability metrics outputs, these metrics are used to build the prediction models to predict stability. The selected metrics are introduced in section 4.3.3.

4.3.2 Metrics correlation

Many software metrics are selected from the previous phase; some metrics may have high degree of correlation. To define the software metrics that can be used to build and predict stability, Minitab tool is used to measure the degree of correlation between metrics using Pearson correlation coefficient measure. The correlation coefficient output value lies between -1 and 1. When two metrics increase together in likeness then the correlation coefficient value is positive. While, if one increase while the other decreases then the correlation coefficient value is negative. Table 4-7 shows an example of the correlation coefficient values between metrics.

TABLE 4-7: Sample of correlation coefficient values between metrics

	V(G)	DIT	NOA	NRM	NLM	WMC
LCOM	0.224	-0.107	0.131	0.244	0.843	0.002
NPM	0.216	-0.187	0.24	0.2	0.942	-0.065
LOC	0.573	0.285	0.51	0.682	0.698	0.42
NOM	0.249	0.055	0.424	0.423	0.969	0.188
WAC	0.392	0.515	1	0.476	0.356	0.593

It can be noticed from Table 4-7 that number of attributes (NOA) metric and weighted attributes per class (WAC) metric increase together as the correlation coefficient value between them is 1. The same rule also applies on a number of local methods (NLM) and number of methods (NOM). Because of this, NOA and NOM metrics are excluded. The list of all selected software metrics is introduced in the next section.

4.3.3 Metrics Selection

Software metrics are selected to build the class stability prediction models. This section presents the selected software metrics; Table 4-8.Appendix C contains the details of the selected software metrics.

TABLE 4-8: Selected metrics

Metric	Description
CBO	Coupling Between Object
DAC	Data Abstraction Coupling
DIT	Depth of Inheritance Tree
LCOM	Lack of Cohesion in Methods
LOC	Lines of Code
NLM	Number of Local Methods
NOA	Number of Attributes
RFC	Response For a Class
V(G)	Cyclomatic Complexity
WMC	Weighted Methods per Class

CHAPTER 5

REGRESSION EXPERIMENT

Many software metrics were proposed to measure the stability of the software classes. This chapter introduces the proposed hypotheses and discusses the regression experiments that were performed to test and validate the proposed hypotheses.

5.1 Hypotheses

This section introduces the hypotheses that were proposed to predict class stability.

CII Hypotheses

Hypothesis: Software metrics are able to predict class stability as measured by CII metric.

Null hypothesis: Software metrics are not able to predict class stability as measured by CII metric.

Grosser Hypotheses

Hypothesis: Software metrics are able to predict class stability as measured by Grosser metric.

Null hypothesis: Software metrics are not able to predict class stability as measured by Grosser metric.

Stab Hypotheses

Hypothesis: Software metrics are able to predict class stability as measured by Stab metric.

Null hypothesis: Software metrics are not able to predict class stability as measured by Stab metric.

CSM Hypotheses

Hypothesis: Software metrics are able to predict class stability as measured by CSM metric.

Null hypothesis: Software metrics are not able to class predict stability as measured by CSM metric.

5.2 Experiment Description

The research objective is to predict the object-oriented class stability using software metrics(the selected software metrics are listed in chapter four). Two AI techniques namely Neural Network (Multilayer Perceptron) and Support Vector Machine (SVM) are selected to build the prediction model using DTREG tool, where as Minitab is used to build Linear Regression model. This section will further describe the parameters that are required to build the prediction models.

5.2.1 DTREG for Neural Network (MLP)

Multilayer Perceptron (MLP) is the selected neural network technique. It is one of the most widely used types of neural networks. DTREG provides Multilayer Perceptron Neural Network (MLP) with different parameters that help to build the prediction model [58]. This section introduces the MLP parameters. Figure 5-1 shows the MLP parameters screen.

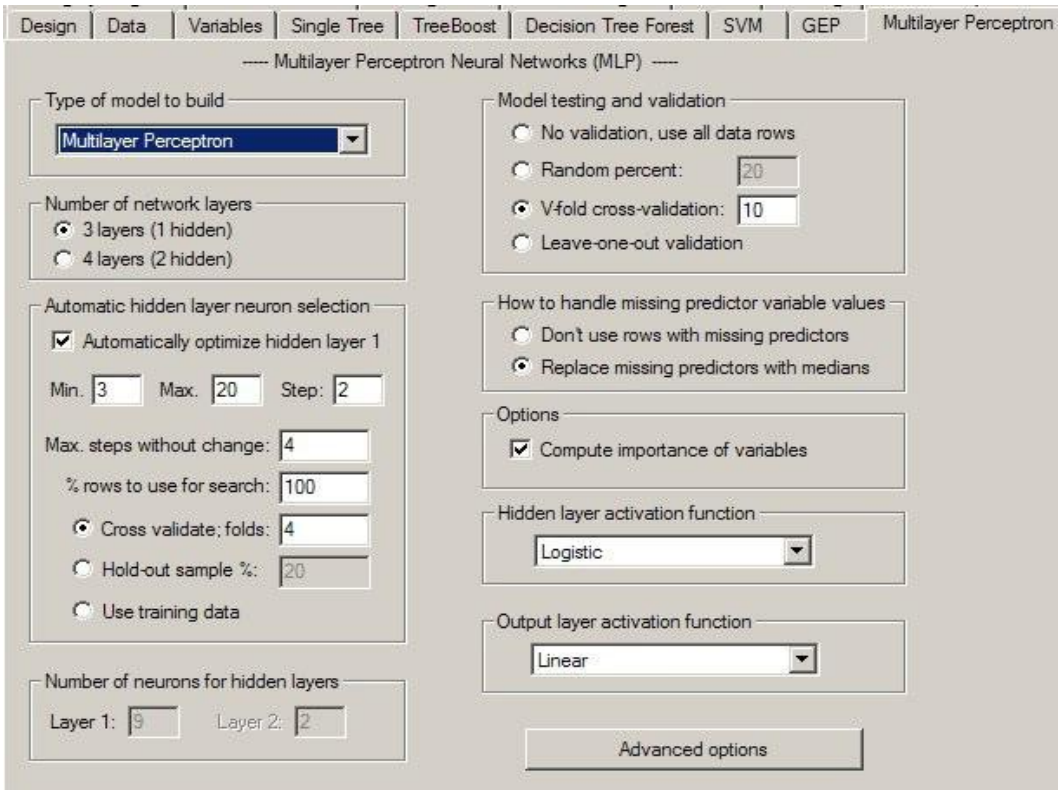


Figure 5-1: MLP parameters screen

Number of network layers: This parameter gives the option of building MLP with either three layers or four layers (out of which two are hidden layers).

Automatic hidden layer neuron selection: Selecting the number of neurons in a hidden layer is a challenging task; this option helps to find the optimal number of neurons and to evaluate the model fitting by using cross validation or a hold-out sample. This option also allows the user to specify the maximum and minimum number of neurons in order to find the optimal numbers of neurons, number of neurons to be added in each trial, and the maximum number of neurons after which the trial should stop if there is no enhancement.

Model testing and validation: After the model is created by DTREG, it can be tested and validated using this option.

Hidden & Output layers activation function: Allows the selection of the activation function such as linear or logistic (sigmoid) to be used in the hidden and output layers.

5.2.2 DTREG for Support Vector Machine (SVM)

Support vector machine (SVM) is an AI technique that is used to analyze data, recognize and classify patterns, and analyze regression. DTREG provides Support vector machine (SVM) with different parameters that helps to build the prediction model[58]. This section introduces the MLP parameters. Figure 5-2 shows the SVM parameters screen.

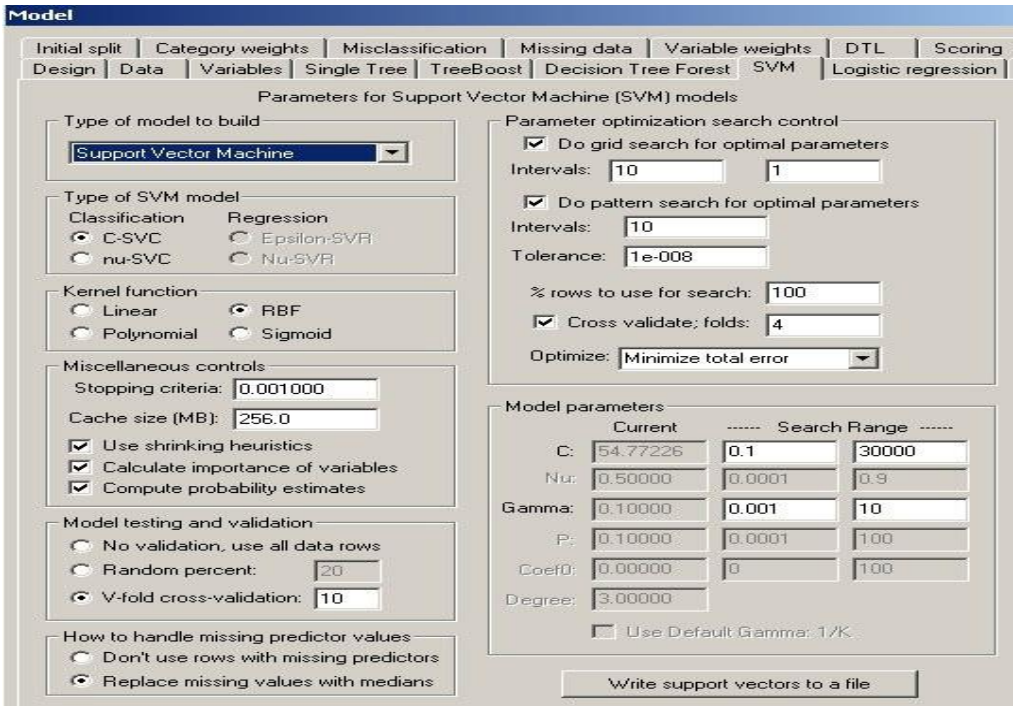


Figure 5-2: SVM parameters screen

Type of model to build: To select the type of model, classification or regression.

Kernel function: The different types of kernel functions offered by DTREG.

Miscellaneous controls: Different parameters control the model, some of these parameters such as stopping criteria are used to control the iterative optimization process i.e. when it should be stopped,, cache size is used to improve the operation performance, shrinking heuristics is used to speed up the process by ignoring points which are unlikely to influence the choice of the optimal separating hyper-plane.

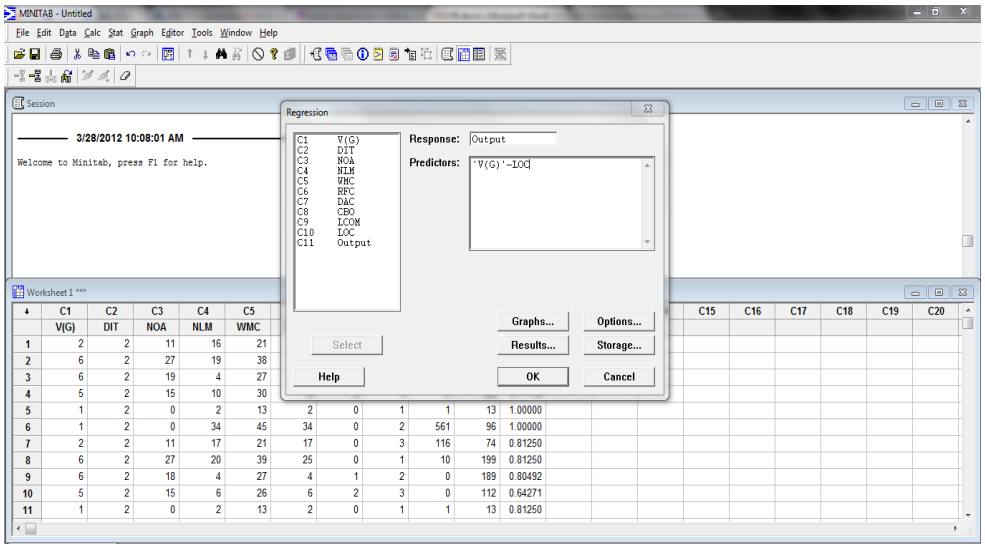
Model testing and validation: Once the model is created by DTREG, it can be tested and validated using this option.

Parameter optimization search control: DTREG provides two different methods that will help to find the optimal parameter values, namely grid and pattern searches.

Model parameters: These parameters are based on the selected kernel functions, they are important as they affect the accuracy of the SVM model.

5.2.3 Minitab for Linear Regression

Minitab tool is used to perform a linear regression analysis to find out the relationship between the target variable and the predictors variables by molding the example data using linear functions. Linear regression analyzes the relationship between the dependent or target variable and the independent variables or predictor variables [65]. Figure 5-3 shows the Minitab linear regression screen.



	C1	C2	C3	C4	C5	C15	C16	C17	C18	C19	C20
	V(G)	DIT	NOA	NLM	WMC						
1	2	2	11	16	21						
2	6	2	27	19	38						
3	6	2	19	4	27						
4	5	2	15	10	30						
5	1	2	0	2	13	2	0	1	1	13	1.00000
6	1	2	0	34	45	34	0	2	561	96	1.00000
7	2	2	11	17	21	17	0	3	116	74	0.81250
8	6	2	27	20	39	25	0	1	10	199	0.81250
9	6	2	18	4	27	4	1	2	0	189	0.80492
10	5	2	15	6	26	6	2	3	0	112	0.64271
11	1	2	0	2	13	2	0	1	1	13	0.81250

Figure 5-3: Minitab linear regression screen

First, data is inserted in order to start the analysis process. The data can be typed directly to the Minitab worksheet or imported from the source. Minitab parameters are:

Response: This selects the dependent "output" variable that needs to be predicted.

Predictors: This selects the independent "input" variables.

5.3 Terminology

This section describes the terminology that is used throughout the following discussions.

Auto: Identify the number of neurons in the hidden layer automatically; select "Automatic hidden layer neuron selection" option in DTREG.

Manual: Identify the number of neurons in the hidden layer manually by user; unselect Automatic hidden layer neuron selection" option in DTREG.

All-projects: Dataset is created from the all selected Java projects

Sx: Experiment setup

Vx: Dataset file version, the data rows are shuffled to generate new version

DS: Dataset

5.4 Experiment Setup

Several setups with different parameters are used in regression experiments to predict the object-oriented class stability based on artificial intelligence techniques. The software metric "Stab" that was proposed by Daniel et al. [66] was excluded from the experiment as the "Stab" metric is more suitable for classification analysis rather than regression analysis that is being done in this experiment. As previously mentioned in chapter four "Data Collection and Analysis", three Java open source projects were selected to be part of the research experiments. Two phases are designed for neural network experiments, the initial phase and the primary phase. The goal of the initial phase is to build neural network models with many different setups using a portion of the dataset to get a first impression of the accuracy of the prediction models and also to verify the neural network parameters for the different setup for the next phase. The NetBeans Java project is selected as the source of the dataset for the initial phase experiment of the neural network. The dataset file that is used to build the prediction models consists of the values of selected software metrics as predictor variables and class stability metric output value as a target variable. Figure 5-4 shows a sample of the dataset file. This section discusses the different setups that are used to build neural network and support vector machine models for regression.

	A	B	C	D	E	F	G	H	I	J	K
1	V(G)	DIT	NOA	NLM	WMC	RFC	DAC	CBO	LCOM	LOC	Output
2	1	2	0	5	16	5	0	1	10	27	35
3	1	2	0	4	15	4	0	1	6	30	11.11111
4	7	2	0	5	24	16	0	13	15	113	276.6667
5	13	2	18	6	29	8	0	2	0	66	6.451613
6	13	2	18	6	29	8	0	2	0	66	0
7	13	2	18	6	29	8	0	2	0	66	0
8	4	2	1	8	23	10	1	3	22	90	2.272727
9	4	2	1	8	23	10	1	3	22	90	0
10	4	2	1	8	23	10	1	3	22	90	0
11	14	2	4	3	32	7	2	9	3	142	8.396947
12	15	2	4	3	33	7	2	9	3	150	5.633803
13	15	2	4	3	33	7	2	9	3	150	0
14	2	2	5	5	17	5	1	2	10	57	50
15	2	2	5	5	17	5	1	2	8	37	35.08772

Figure 5-4: Sample of the dataset file

5.4.1 Neural Network (MLP) Initial Phase Setup

Ten different setups are proposed to be part of the regression experiment. These setups are based on the parameters of neural network and genetic algorithm, this section will describes these parameters. Two setup types are selected to be part of each setup, “Auto” type which uses DTREG option to find the optimal number of neurons in the hidden layer automatically, and “Manual” type which is used to set the number of neurons in the hidden layer manually. CII, Grosser, and CSM class stability metrics will be validated using these setups. Cross-validation with 10 folds is used to evaluate the quality and accuracy of the neural network prediction model. The setup parameters that are used to build different models are follows:

- Number of neurons in the hidden layer
- Minimum and maximum number of neurons in the hidden layer
- Number of neurons added in each step
- Number of steps after which the trial should stop (if no change)
- Activation function type
- Maximum generation
- Mutation value
- Crossover value

Table 5-1 lists of the setup parameters that are used with each type.

TABLE 5-1: Experiment setup parameters

Setup	Setup Type	
	Auto	**Manual**
1	DTREG default setup	DTREG default setup
2	Max neurons = 50	Number of neurons in the hidden layers = 10
3	Max generation = 100	Number of neurons in the hidden layers = 6 Max generation = 100
4	Max neurons = 20	Number of neurons for hidden layers = 10 Max generation = 100
5	Mutation = 0.1 Crossover = 0.2	Number of neurons in the hidden layers = 6 Mutation = 0.1 - Crossover = 0.2
6	Mutation = 0.1 Crossover = 0.2 Max generation = 100	Number of neurons in the hidden layers = 6 Mutation = 0.1 - Crossover = 0.2 Max generation = 100
7	Mutation = 0.6 Crossover = 0.2	Number of neurons in the hidden layers = 6 Mutation = 0.6 - Crossover = 0.2
8	Mutation = 0.2 Crossover = 0.6	Number of neurons in the hidden layers = 6 Mutation = 0.2 - Crossover = 0.6
9	Output Activation function = logistic	Number of neurons for hidden layers = 4
10	Number of neurons in the second hidden layers = 2	Number of neurons in the second hidden layers = 2

5.4.2 Neural Network (MLP) Primary Phase Setup

The variance results of the initial phase experiment were almost similar in the most setups which did not help in selecting the appropriate setup for the primary phase. Therefore, three setups are selected considering genetic algorithm parameters such as crossover and mutation. Each setup performs five experiment runs and each run uses different dataset file version. The "Auto" option is used with the first experiment run in each setup to define the number of neurons in the hidden layer which will be used in the next experiment runs. Table 5-2 shows the primary phase setup parameters.

TABLE5-2: Primary phase setup parameters

Setup	Setup parameter
1	Default DTREG setting Mutation = 0.75 Crossover = 0.6
2	Mutation = 0.1 Crossover = 0.6
3	Mutation = 0.6 Crossover = 0.1

The selected Java projects are used in this phase to build four datasets. A dataset from each projects and one dataset represents all projects. Hold-out sample validation method

is used to validate the neural network models as the data is divided into 70% for training and 30% for validation.

5.4.3 Support Vector Machine (SVM) Setup

DTREG provides SVM with two regression models, Epsilon-SVR and Nu-SVR. The two methods solve the same optimization problem using different parameters. The kernel function is the most important parameter that is used to build the SVM model. DTREG offers different kernel functions. Three functions are selected to be part of the SVM regression experiment, linear, Radial Basis Function (RBF), and sigmoid kernel functions. All selected Java projects are used to create four datasets for SVM regression experiment. A dataset from each projects and one dataset represents all projects. Two validation methods are used: cross-validation with 10 folds to validate models during the optimization search process, and hold-out sample with 70% for training and 30% for validation to evaluate and test the final SVM model. The SVM experiment is based on the regression methods and different kernel functions. Therefore, one experiment phase was designed for the SVM. Figure 5-3 shows the SVM setups.

TABLE 5-3: SVM setups

Setup	Regression Method	
	Epsilon-SVR	**Nu-SVR**
1	Kernel function = Linear	Kernel function = Linear
2	Kernel function = RBF	Kernel function = RBF
3	Kernel function = Sigmoid	Kernel function = Sigmoid

5.4.4 Multiple Linear Regression (LR) Setup

Linear regression models are simple to implement and model training the model is usually much faster than other regression methods such as neural networks. All selected Java projects dataset is used in this experiment. Data is inserted directly to the Minitab worksheet.

5.5 Experiment Results

The regression experiment is conducted to predict class stability. Several techniques are used in the experiments and different estimators are considered based on the used techniques such as mean square error (MSE) and R-square. They measure the quality of the prediction model and how well the future outcomes are likely to be predicted by the model. This section shows and discusses the experiment results of the different techniques. Table 5-4 describes the estimators.

TABLE 5-4: Experiment estimators

Estimators	Description
Mean Square Error (MSE)	Estimates the difference between the predicted value and the actual one. It calculates the average of the square errors of the difference that occurs because of randomness. MSE value should not be exceeded 20% to 25% error percentage.
R-square	Measures how well a regression model approximates real data points by measuring the square of the correlation coefficient between the outputs and predicted values. R-square should be above the 80%.
P-Value	The probability to reject null hypothesis that no relation between predictor variables and output. A commonly accepted value is 0.05.

To measure the accuracy of the models that are created by the neural network technique, Mean Square Error (MSE) is used for models that are created for CSM and Grosser metrics, Normalized Mean Square Error (NMSE) is used for the model that is created for the CII metric as its output is not normalized. The result sections show MSE and NMSE as values. They are then converted to error percentage by multiplying the error value by 100 during the discussion. R-square and P-value are used for the LR models in all stability metrics. This section presents the results of prediction models.

5.5.1 Neural Network (MLP) Initial Phase Results

The NetBeans project is selected to conduct the initial phase experiment. This section presents the experiment results of the neural network (MLP) initial phase for class stability metrics and the observations about the results.

5.5.1.1 MLP Initial Phase Results for the CII Metric

Neural network (MLP) models that are created to validate the CII metric generate unacceptable error. Table 5-5 presents the results of the MLP initial phase.

TABLE 5-5: MLP initial phase results for CII metric

CII metric										
NMSE	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**
M	3.39	3.39	3.32	3.59	1.22	5.95	1.3	1.39	1.07	94.8
A	1.05	1.05	1.13	14.97	21.03	2.56	1.86	1.47	1.07	1.01

Although, many setups have been tested, the minimum error is still over 100%. Some of the experiments even gave aberrant error values, for instance the error value of Setup 10 under the “Manual” type. The same applies to Setup 4 and 5 under the “Auto” type.

5.5.1.2 MLP Initial Phase Results for the Grosser Metric

On the other hand, neural network (MLP) prediction models that are created for the Grosser metric generate good results. Table 5-6 presents the MLP initial phase experiment results for the Grosser metric.

TABLE 5-6: MLP initial phase results for Grosser metric

Grosser metric										
MSE	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**
M	0.13	0.13	0.02	0.03	0.01	0.01	0.17	0.01	0.15	0.01
A	0.01	0.01	0.04	0.02	0.01	0.01	0.01	0.15	0.01	0.01

1% is the minimum error even though there are some experiments which generate considerably larger errors than the minimum, for example, Setup 7 under the "Manual" type generates 17% error, the highest error in this experiment.

5.5.1.3 MLP Initial Phase Results for the CSM Metric

Neural network (MLP) prediction models for the CSM metric in the initial phase show the good performance of the models with a minimum error of 2%. Table 5-7 presents the MLP initial phase experiment results for the CSM metric.

TABLE 5-7: MLP initial phase results for CSM metric

CSM metric										
MSE	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**
M	0.02	0.02	0.04	0.04	0.03	0.03	0.02	0.06	0.02	0.02
A	0.02	0.02	0.04	0.03	0.07	0.03	0.02	0.02	0.02	0.02

The variance between the minimum and the maximum error percentage is low as it ranges between 2% to 7%.

5.5.1.4 Observations about MLP Initial Phase Results

Some key observations about the MLP Initial phase results are as follows:

- The CII metric cannot be used as predictor for class stability using the selected software metrics as CII measures class stability based on the percentage of the lines of code changed between versions. The output of CII may not be a good factor to be used for stability prediction.
- The experimental results show low error values for both Grosser and CSM metrics, therefore, they can be used as a predictor for class stability.
- The variance results of the initial phase experiments were almost similar for the most setups which do not helping in selecting the appropriate setup for the primary phase.

5.5.2 Neural Network (MLP) Primary Phase Results

Three setups are selected considering genetic algorithm parameters. This experiment was conducted on the four datasets which were created using the all selected Java projects. This section presents the experiment results of the MLP primary phase for class stability metrics and the observation about the results.

5.5.2.1 MLP Primary Phase Results for the CII Metric

Table 5-8 shows the regression experiment results of neural network (MLP) models for the CII metric.

TABLE 5-8: MLP primary phase results for the CII metric

CII metric												
DS	**Android**			**Eclipse**			**NetBeans**			**All-projects**		
	S1	**S2**	**S3**	**S1**	**S2**	**S3**	**S1**	**S2**	**S3**	**S1**	**S2**	**S3**
V1	0.7049	0.9125	0.9141	0.953	0.9884	0.9991	1.417	1.201	1.432	1.0996	0.9993	1.1511
V2	0.9253	0.9265	0.9532	1.026	1.0235	4.0596	1.079	0.739	1.320	1.1445	1.1880	1.2722
V3	0.9312	0.917	0.8872	1.009	101.21	9.1755	1.149	27.50	1.053	7.3094	0.9737	1.3132
V4	1.3134	1.0902	0.9274	1.352	1.4678	2.2354	1.036	42.83	1.069	1.2562	1.1490	1.1640
V5	0.9875	0.9233	0.9881	0.978	1.0241	1.2235	1.057	26.52	1.074	1.1367	1.1557	1.0101

- **Android**: Neural network (MLP) models generate high and unacceptable error values, although, the error is reduced to 70% in Setup 1 with dataset file Version 1, this value is larger than the acceptable error percentage of MSE.
- **Eclipse**: The error in this project is also very large. Moreover, Setup 2 with the dataset file Version 3 shows abnormal values compared to other results.
- **NetBeans**: The same results as the MLP initial phase experiment.
- **All-projects**: All the selected Java projects data are collected into one dataset. The neural network (MLP) models still generate large and unacceptable errors in all setups.

5.5.2.2 MLP Primary Phase Results for the Grosser Metric

Table 5-9 shows the regression experiment results of neural network (MLP) models for the Grosser metric.

TABLE 5-9: MLP primary phase results for the Grosser metric

Grosser metric												
DS	**Android**			**Eclipse**			**NetBeans**			**All-projects**		
	S1	**S2**	**S3**	**S1**	**S2**	**S3**	**S1**	**S2**	**S3**	**S1**	**S2**	**S3**
V1	0.0058	0.0067	0.0059	0.043	0.0666	0.0613	0.013	0.014	0.013	0.0173	0.0182	0.0181
V2	0.0403	0.0367	0.0056	2.125	0.2894	0.0886	0.015	0.014	0.223	0.0354	0.0331	0.0243
V3	0.0071	0.0053	0.0055	0.219	0.8306	0.7553	0.018	0.019	0.017	0.0343	0.0680	0.0330
V4	0.0073	0.0272	0.0073	0.053	0.0732	0.0950	0.014	0.014	0.014	0.0357	0.0323	0.0321
V5	0.1552	0.0086	0.0616	0.043	0.0503	0.1397	0.022	0.670	0.041	0.0203	0.0926	0.0496

- **Android**: Neural network (MLP) prediction models generate good results. The minimum error is 0.5% which is in the acceptable range, whereas the maximum error is 15%. Setup 2 and Setup 3 generate low error values unlike Setup 1 that generates high error value in Version 5 compared to other setups in the same project.
- **Eclipse**: The minimum error is 4%. All experiments generate acceptable error values but there are some which produce relatively high error, such as Setup 1with Version 2produces high error.
- **NetBeans**: The same results as the MLP initial phase experiment.
- **All-projects**: All the selected Java projects data are collected together into one dataset. The neural network (MLP) models that are built to predict stability generate an acceptable error value for all setup experiments. The minimum error is 1.7% and the highest is 9%. No aberrant value is noticed in this experiment.

5.5.2.3 MLP Primary Phase Results for the CSM Metric

Table 5-10 shows the regression experiment results of neural network (MLP) models for the CSM metric.

TABLE 5-10: MLP primary phase results for the CSM metric

CSM metric												
DS	**Android**			**Eclipse**			**NetBeans**			**All-projects**		
	S1	**S2**	**S3**	**S1**	**S2**	**S3**	**S1**	**S2**	**S3**	**S1**	**S2**	**S3**
V1	0.0178	0.0182	0.0172	0.046	0.0509	0.0462	0.028	0.034	0.030	0.0326	0.0331	0.0317
V2	0.0176	0.0286	0.0189	0.055	0.0604	0.0579	0.042	0.032	0.026	0.0334	0.0336	0.0292
V3	0.0247	0.0182	0.0203	0.059	0.0644	0.0484	0.030	0.028	0.030	0.0351	0.0351	0.0327
V4	0.0208	0.0175	0.0242	0.046	0.0511	0.0499	0.040	0.032	0.020	0.0311	0.0310	0.0331
V5	0.0330	0.0169	0.0296	0.052	0.0555	0.0499	0.029	0.035	0.032	0.0309	0.0310	0.0321

- **Android**: The experiment results show that neural network (MLP) models that are built to predict stability using the CSM metric are fairly accurate as the minimum error is 1.7% and all setups with different versions generate acceptable error values.
- **Eclipse**: The experiment used in this project also gave acceptable error results. The minimum error is 4.6% which is higher than the one obtained from the Android project.
- **NetBeans**: The same results as the MLP initial phase experiment.
- **All-projects**: Data from all Java projects is used to build neural network (MLP) models. The results show that the models are fairly accurate with the minimum error being 2.9% which is an acceptable value. In addition, the results from different setups with different dataset file versions are all acceptable and no aberrant value is noticed.

5.5.2.4 Observations about MLP Primary Phase Results

Some key observations about the MLP Primary Phase results are:

- Grosser and CSM prediction model results show that these metrics can be used as predictor for class stability using the selected software metrics.
- The lowest errors were obtained from the Android project for the two metrics: Grosser metric with0.5% and CSM metric with1.7%. Most of the Android classes are absolutely stable based on the Grosser and CSM measurement results. This might explain the reason why the neural network (MLP) models are able to predict class stability and result with the lowest error values compared to other projects.

- Neural network (MLP) models for the Grosser metric gave acceptable error values except for some experiments they gave aberrant values. As the Grosser metric is designed to measure class stability based only on one factor which is number of method "NOM". The prediction model will be sensitive to this factor which may affect its accuracy.
- On the other hand, the low and high error values of neural network (MLP) models for the CSM metric are all acceptable and there is no abnormal value. Since the CSM metric uses more factors to measure class stability, it is less sensitive to changes in any one factor. The factors which are used in the CSM metric to measure stability reflect the selected software metrics.
- Neural network (MLP) models for the Grosser metric generate the lowest error value in the Android project. This is because most of the Android project classes have an absolute stability according to the Grosser metric.
- If we compare the minimum error of the initial phase which is 1% to the minimum error of the primary phase which is 2.9%, there is no significant difference for all projects in primary phase. An increase in error of 1.9% is observed, this is due to the different datasets that were involved in the prediction process. However, the error value is still acceptable and this shows that neural network using Grosser and CSM stability metrics can be used to build a good quality prediction models for stability using the selected software metrics.

5.5.3 Support Vector Machine (SVM) Results

Two regression methods are provided by DTREG to build the SVM model. Three kernel functions were used in SVM regression experiment. The Normalized Mean Square Error (NMSE) is used to measure the accuracy of the SVM prediction models for the CII metric, while the Mean Square Error (MSE) is used for models of the Grosser and CSM metrics. This section presents the results of SVM experiments for the class stability metrics.

5.5.3.1 SVM Results for the CII Metric

Table 5-11 shows the regression experiment results of the SVM models for the CII metric.

TABLE 5-11: SVM results for the CII metric

CII metric						
Android project						
DS	**Epsilon-SVR**			**Nu-SVR**		
	Linear	**RBF**	**Sigmoid**	**Linear**	**RBF**	**Sigmoid**
V1	1.005	0.433	0.955	0.993	0.437	1.1254
V2	1.005	0.644	1.306	1.0005	0.923	1.262
V3	1.005	0.361	1.453	1.006	0.468	1.296
Eclipse project						
V1	1.007	1.007	1.016	1.007	1.006	1.026
V2	1.007	1.032	1.018	1.007	1.006	1.017
V3	1.007	1.004	1.014	1.007	1.007	1.034
NetBeans project						
V1	1.005	1.008	1.001	1.005	1.003	0.998
V2	1.005	1.005	0.996	1.005	1.005	1.019
V3	1.005	1.005	1.004	1.005	1.005	1.011
All-projects						
V1	1.002	1.002	0.999	1.002	1.002	1
V2	1.002	1.001	1.002	1.002	1.002	1.004
V3	1.003	0.998	1	1.004	1.006	1.003

- **Android**: The error is reduced to 36% in this project compared to the same project in the neural network. The error is still high and unacceptable.
- **Eclipse**: High error is generated by the SVM models for the CII metric.
- **NetBeans**: High error is generated by the SVM models for the CII metric.
- **All-projects**: Different SVM regression methods with various kernel functions are used; still the SVM models for the CII metric generate high error values.

5.5.3.2 SVM Results for the Grosser Metric

SVM models that are created to predict the class stability using the Grosser metric generate good results. Table 5-12 shows the regression experiment results of the SVM models for the Grosser metric.

TABLE 5-12: SVM results for the Grosser metric

Grosser metric						
Android project						
DS	**Epsilon-SVR**			**Nu-SVR**		
	Linear	**RBF**	**Sigmoid**	**Linear**	**RBF**	**Sigmoid**
V1	0.0054	0.0054	0.0051	0.005	0.005	0.005
V2	0.0054	0.0054	0.005	0.005	0.005	0.005
V3	0.0054	0.0054	0.005	0.005	0.008	0.005
Eclipse project						
V1	0.0344	0.034	0.0345	0.034	0.034	0.034
V2	0.0342	0.039	0.034	0.034	0.034	0.037
V3	0.0342	0.034	0.034	0.034	0.034	0.034
NetBeans project						
V1	0.014	0.014	0.014	0.014	0.015	0.014
V2	0.014	0.014	0.013	0.013	0.014	0.014
V3	0.014	0.014	0.015	0.014	0.013	0.014
All-projects						
V1	0.017	0.017	0.017	0.017	0.017	0.017
V2	0.017	0.017	0.017	0.0169	0.017	0.017
V3	0.017	0.017	0.017	0.017	0.018	0.017

- **Android**: The error is 0.5% in all experiments except one experiment under Nu-SVR method which is 0.8%. This shows that the prediction models are fairly accurate.
- **Eclipse**: The error is increased in this project to 3% but is still acceptable. Unlike the Android projects, the error is almost the same in all experiments.
- **NetBeans**: The SVM models for the Grosser metric perform the same as the SVM models in the Eclipse project. The error is 1% in all experiments.
- **All-projects**: Finally the SVM models for All-projects also show the good performance and the error is 1.7% with no abnormal noticed.

5.5.3.3 SVM Results for the CSM Metric

Once more, the SVM models for the CSM metric show the accuracy of the models base in the experiment results. Table 5-13 shows the regression experiment results of the SVM models for the CSM metric.

TABLE 5-13: SVM results for the CSM metric

CSM metric						
Android project						
DS	**Epsilon-SVR**			**Nu-SVR**		
	Linear	**RBF**	**Sigmoid**	**Linear**	**RBF**	**Sigmoid**
V1	0.0157	0.0157	0.0169	0.0152	0.0153	0.0157
V2	0.0157	0.0168	0.0167	0.0153	0.0152	0.0159
V3	0.0157	0.0167	0.0159	0.0153	0.0154	0.0174
Eclipse project						
V1	0.05	0.05	0.051	0.048	0.047	0.055
V2	0.049	0.046	0.052	0.048	0.047	0.05
V3	0.047	0.047	0.049	0.048	0.0467	0.0522
NetBeans project						
V1	0.027	0.037	0.03	0.027	0.036	0.028
V2	0.027	0.031	0.027	0.027	0.029	0.028
V3	0.027	0.031	0.028	0.027	0.033	0.027
All-projects						
V1	0.032	0.032	0.031	0.032	0.031	0.032
V2	0.029	0.032	0.032	0.031	0.03	0.03
V3	0.032	0.03	0.031	0.031	0.031	0.034

- **Android**: The error is 1% and it is the same in all experiments. All kernel functions generated the same results.
- **Eclipse**: The error increases in this project up to 4.6% compared to the previous project. However, the error still acceptable and proves that the SVM models are of good quality.
- **NetBeans**: The SVM models perform as expected in this project and produce 2.7% error
- **All-projects**: The SVM models generate good results and the error is 3% which shows the accuracy of the models.

5.5.3.4 Observations about SVM Results

Some key observations about the SVM results are:

- The SVM models for Grosser and CSM metrics can be used to build a good quality prediction models to predict class stability using the selected software metrics.
- Building SVM models is time consuming unlike the neural network (MLP) models. However, the results are the same in almost all experiments using Grosser or CSM metrics and no abnormal error value has been noticed. This is because the search method that is used by SVM tries to find the region near the global minimum before starting the search process.

5.5.4 Multiple Linear Regression Results

Linear regression is used by many researchers to predict software quality. Therefore, an experiment is designed and performed using linear regression on class stability metrics to observe the results. Two estimators were used to validate the LR results, R-squared and P-value. This section presents the LR experiment results using class stability metrics.

5.5.4.1 LR Results for the CII Metric

In general, the R-squared results are lower than the acceptable percentage for the CII metric (The higher the R-squared value, the better the model fits the data). Table 5-14 presents the LR results for the CII metric. Although, the P-value of LR model in Android and All-projects are less than the common value "0.05", but still the R-squared values that represents the percentage of total variation in the target variable as explained by predictors are not acceptable.

TABLE 5-14: LR results for the CII metric

CII metric		
Project	**R-Sq**	**P-Value**
Android	27.6%	0
Eclipse	3.30%	0.537
NetBeans	2.10%	0.855
All-projects	3.60%	0.002

5.5.4.2 LR Results for the Grosser Metric

Again, the R-squared results show low percentage of multi linear regression model for the Grosser metric. Table 5-15 presents the LR results for the Grosser metric. The higher the R-squared, the better the model fits the data. This time, All-projects experiment has acceptable P-value whereas other projects show the P-value greater than common value.

TABLE 5-15: LR results for the Grosser metric

Grosser metric		
Project	**R-Sq**	**P-Value**
Android	6.90%	0.106
Eclipse	6.30%	0.064
NetBeans	4.40%	0.312
All-projects	4.50%	0

5.5.4.3 LR Results for the CSM Metric

Finally, the multi linear regression for the CSM metric also shows low R-squared percentage like other stability metrics. Table 5-16 presents the LR results for the CSM metric. Also the all-project experiment gives acceptable P-value in this experiment.

TABLE 5-16: LR results for the CSM metric

CSM metric		
Project	**R-Sq**	**P-Value**
Android	1.80%	0.95
Eclipse	6.10%	0.077
NetBeans	3.10%	0.622
All-projects	4.50%	0

5.5.4.4 Observations on LR Results

- The R-squared values are low for all stability metrics. As a result, the predictor variables do not accurately predicts the target variable using multi linear regression.
- Some experiments generate acceptable P-value. However, the R-squared values of these experiments are very low and unacceptable.
- Linear regression builds model based on statistical relationship between software metrics and quality factor. However, the previous studies on software quality prediction showed that the relationship between software metrics and quality factor is complex limiting the accuracy of the traditional approaches. Hence, multi linear regression may not be the good choice to predict class stability using software metrics as measured by class stability metrics.

5.6 Discussion

Regression experiments were conducted using different techniques to build models for class stability prediction using the selected software metrics. This chapter presented many experiment results for neural network (MLP), support vector machine (SVM), and multi linear regression (LR). The prediction models' accuracy was measured by the error which results from the difference between the predicted value and the actual one, this chapter presented the minimum error value that was obtained from each experiment in sections 5.5. In order to evaluate the accuracy of prediction models, the average and standard deviation are calculated to ensure the accuracy of the prediction models. This section presents and discusses the average and standard deviation results for neural network (MLP) and support vector machine (SVM) experiments.

5.6.1 Neural Network (MLP)

The primary phase experiment results were used to calculate the average and standard deviation as all Java projects were involved in this experiment. Table 5-17 shows the results of neural network primary phase models.

TABLE 5-17: MLP primary phase Avg & Std results

Neural Network - MLP						
Stability Metrics	**Average**			**Standard Deviation**		
	Android	**Eclipse**	**NetBeans**	**Android**	**Eclipse**	**NetBeans**
CII	0.953	8.582	7.366	0.126	25.715	13.353
Grosser	0.025	0.329	0.075	0.039	0.557	0.173
CSM	0.021	0.053	0.031	0.005	0.005	0.005

- **CII metric**: As discussed in section 5.5 that the error values of the prediction models for the CII metric are high and not acceptable. As a result, the average and standard deviation of neural network (MLP) models that used CII metric to predict class stability are high and not acceptable.
- **Grosser metric**: It shows good results in most the neural network experiments. However, some aberrant error values were noticed. As a result, the accuracy of prediction models is reduced. For instance, the average and standard deviation values are high and unacceptable in the Eclipse project. While the average values in NetBeans project are acceptable but high standard deviation makes the error upper bound unacceptable. Models for Android project show good average and good standard deviation. This is because of the absolute stability of most of its classes.
- **CSM metric**: Neural network (MLP) models for CSM metric give the best average and standard deviation results compared to CII and Grosser metrics. CSM metric uses many factors to measure class stability which makes the prediction models to be

less sensitive to changes and as the result, the accuracy of the models is increased. The best average and standard deviation of models were obtained from the Android project followed by the NetBeans project and then finally the Eclipse project. Figure 5-5 illustrates the neural network models performance for Grosser and CSM, CII is excluded as it generates unacceptable error in the all experiments.

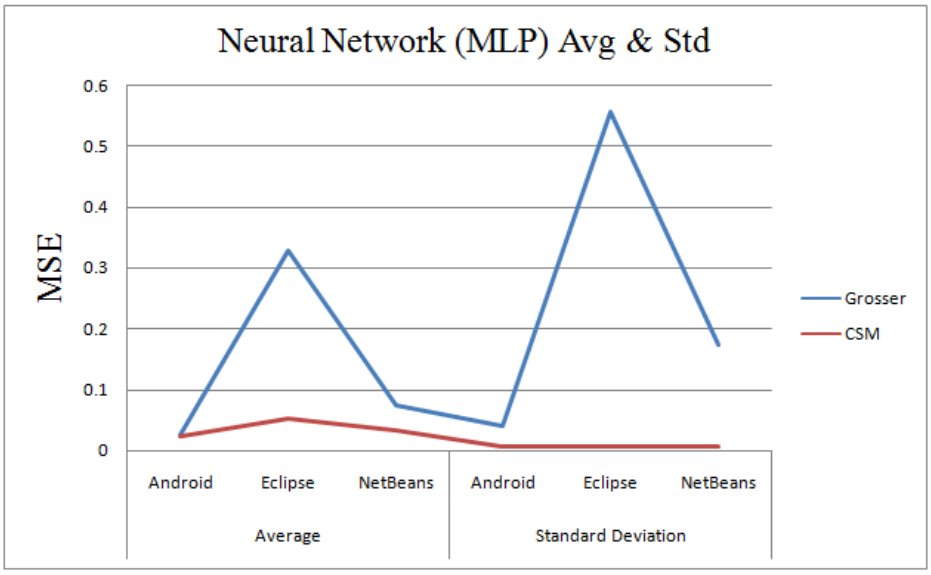


Figure 5-5: MLP models performance

5.6.2 Support Vector Machine (SVM)

The average and standard deviation are calculated for all SVM experiments in the selected Java projects. Table 5-18 shows the results of support vector machine models.

TABLE 5-18: SVM Avg & Std results

Support Vector Machine - SVM						
Stability Metrics	**Average**			**Standard Deviation**		
	Android	**Eclipse**	**NetBeans**	**Android**	**Eclipse**	**NetBeans**
CII	0.927	1.013	1.005	0.328	0.0093	0.0049
Grosser	0.0056	0.034	0.014	0.0006	0.0015	0.0004
CSM	0.0159	0.049	0.029	0.0006	0.0023	0.0033

- **CII metric**: The SVM prediction models for the CII metric generate high and unacceptable error values as discussed in section 5.5. As a result, the average and standard deviation of support vector machine models that use the CII metric to predict stability are also high and not acceptable.
- **Grosser metric**: The average and standard deviation of error values of SVM models are acceptable in all java projects. As a result, the SVM models for the Grosser metric perform better than that of the neural network models.
- **CSM metric**: The average and standard deviation values for the SVM models are relatively low and good for predicting stability using the CSM metric in all Java projects. The SVM models average and standard deviation values are almost as same as the neural network models values.

Figure 5-6 illustrates the support vector machine models' performance for Grosser and CSM, CII is excluded as it generates unacceptable error in the all experiments.

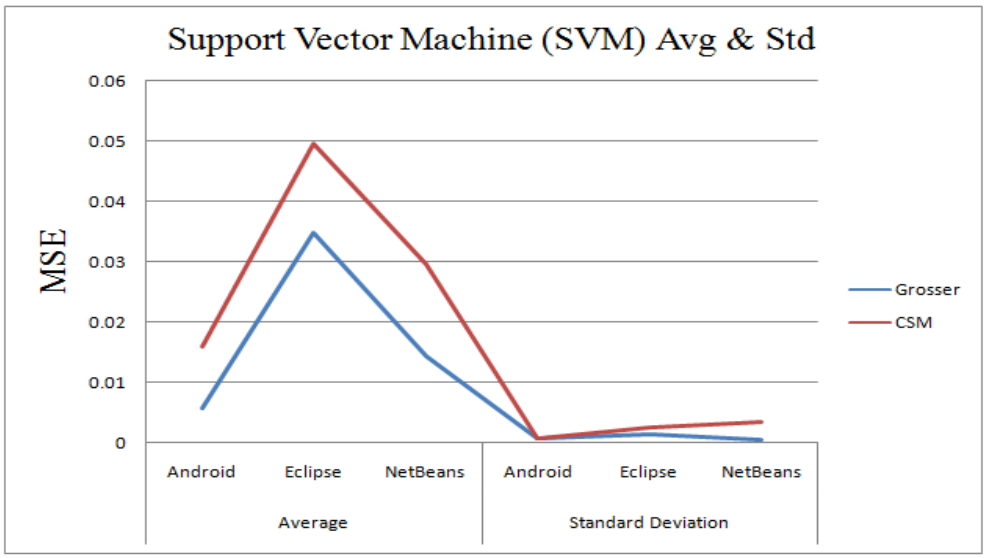


Figure 5-6: SVM models performance

5.6.3 Multi Linear Regression (LR)

The experiment results of all Java projects show that the multi linear regression is not able to predict class stability using the selected software metrics.

5.6.4 Overall

Java projects were collected together to create one dataset "All-projects" to perform the regression experiments using different techniques. In order to evaluate the accuracy of the prediction models, the average and standard deviation are calculated and the results present in Table 5-19.

TABLE 5-19: Regression overall Avg & Std results

Stability Metrics	Neural Network – MLP All-projects		Support Vector Machine – SVM All-projects	
	Average	Standard Deviation	Average	Standard Deviation
CII	1.55489	1.59486	1.002206	0.001799
Grosser	0.03633	0.0204	0.017168	0.00025
CSM	0.03244	0.00163	0.031712	0.00106

- The CII metric cannot be used as a predictor for class stability using the selected software metrics, where as Grosser and CSM metrics can be used as predictor for class stability using the selected software metrics.
- Prediction models for the CSM metric give almost same average and standard deviation values and are both acceptable.
- The CSM metric measure stability based on a number of factors which represent different class properties such as coupling, cohesion, and complexity. Therefore, the CSM metrics reflects the selected software metrics. As a result, the prediction models using the CSM metric are accurate using different AI tarnishes.
- Grosser and CSM metrics accept the proposed hypotheses, where the CII metric reject it.

  Hypothesis: Software metrics are able to predict class stability as measured by class stability metrics.

Figure 5-7 illustrates models performance for Grosser and CSM in All-projects dataset, CII is excluded as it generates unacceptable error in the all experiments.

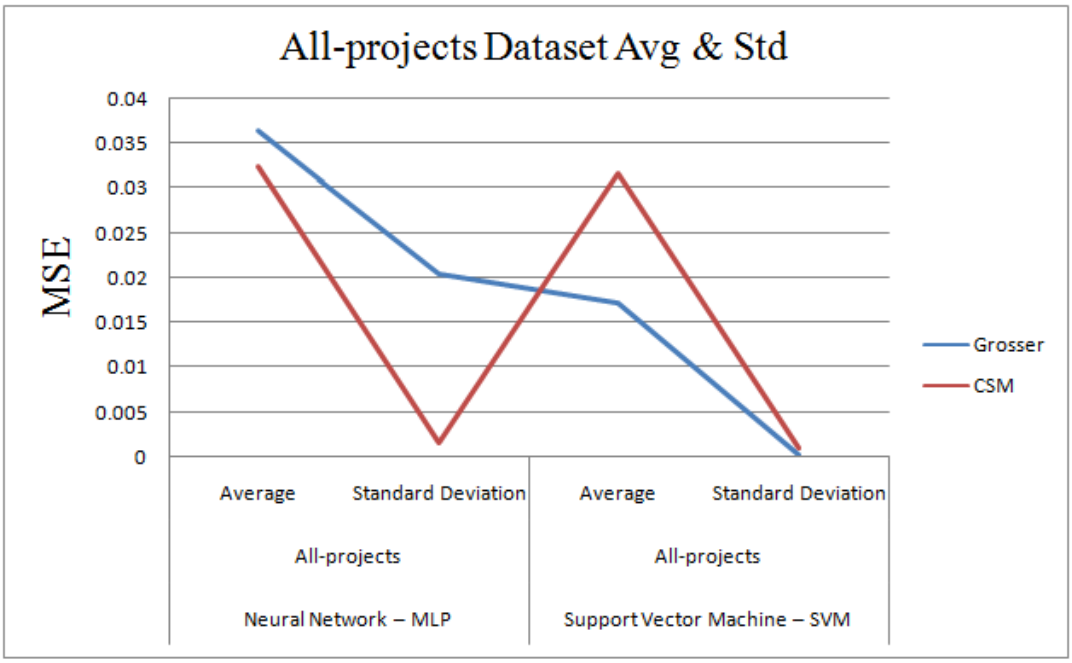


Figure 5-7: Overall models performance

CHAPTER 6

CLASSIFICATION EXPERIMENT

The target value which represents the output of the class stability metrics can be used as a categorical variable 0 or 1, where 0 represents unstable and 1 represents stable. Therefore, the classification experiment is designed to predict either stable or unstable using the selected software metrics which were defined in chapter four. Three stability metrics are involved in this classification experiment, Stab, Grosser, and CSM metrics. The CII metric is excluded as its output cannot be classified as a categorical variable. A threshold value of "0.7" is used based on the result previously obtained from previous studies on stability. This threshold value is used with Grosser and CSM metrics to convert their output values to category value of0 or 1. No conversion process is needed for the Stab metric as its output value is already, either0 or 1. This chapter discusses the classification experiment that was performed to test and validate the proposed hypotheses.

6.1 Hypotheses

This section introduces the hypotheses that were proposed to predict class stability.

CII Hypotheses

Hypothesis: Software metrics are able to predict class stability as measured by CII metric.

Null hypothesis: Software metrics are not able to predict class stability as measured by CII metric.

Grosser Hypotheses

Hypothesis: Software metrics are able to predict class stability as measured by Grosser metric.

Null hypothesis: Software metrics are not able to predict class stability as measured by Grosser metric.

Stab Hypotheses

Hypothesis: Software metrics are able to predict class stability as measured by Stab metric.

Null hypothesis: Software metrics are not able to predict class stability as measured by Stab metric.

CSM Hypotheses

Hypothesis: Software metrics are able to predict class stability as measured by CSM metric.

Null hypothesis: Software metrics are not able to predict class stability as measured by CSM metric.

6.2 Experiment Description

DTREG is used in this experiment to build the classification prediction models for the neural network (MLP) and the support vector machine (SVM). The model parameters were introduced in section 5.2 in the previous chapter for both techniques. Minitab is used to build Logistic Regression model. The only difference that the target variable that should be selected as a categorical variable as shown in Figure 6-1 for neural network and support vector machine models and Figure 6-2 for logistic regression.

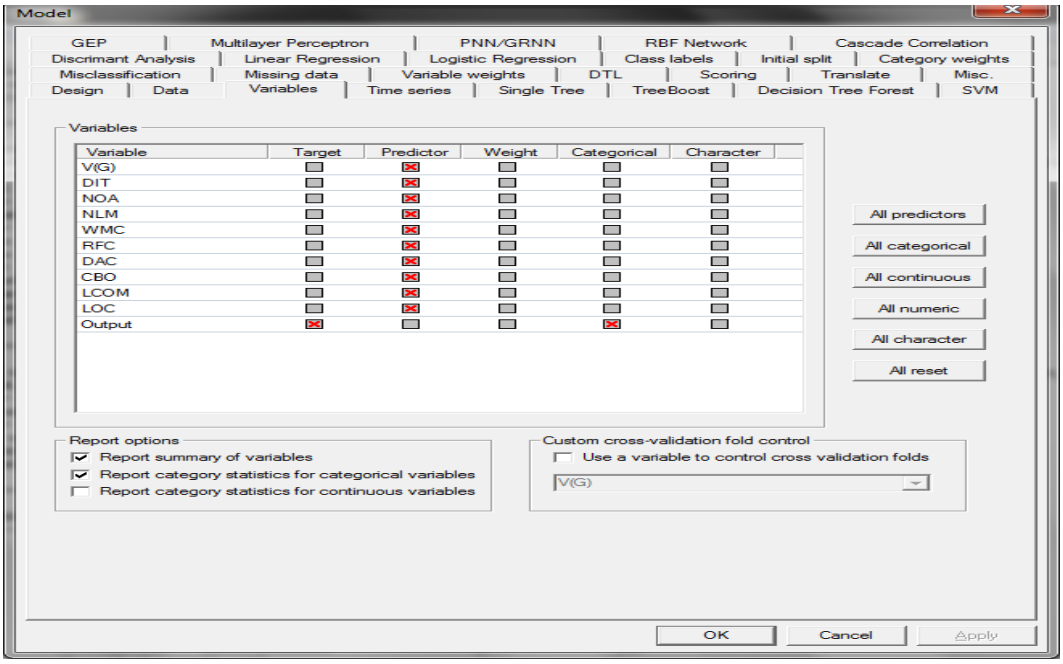


Figure 6-1: DTREG categorical variable screen

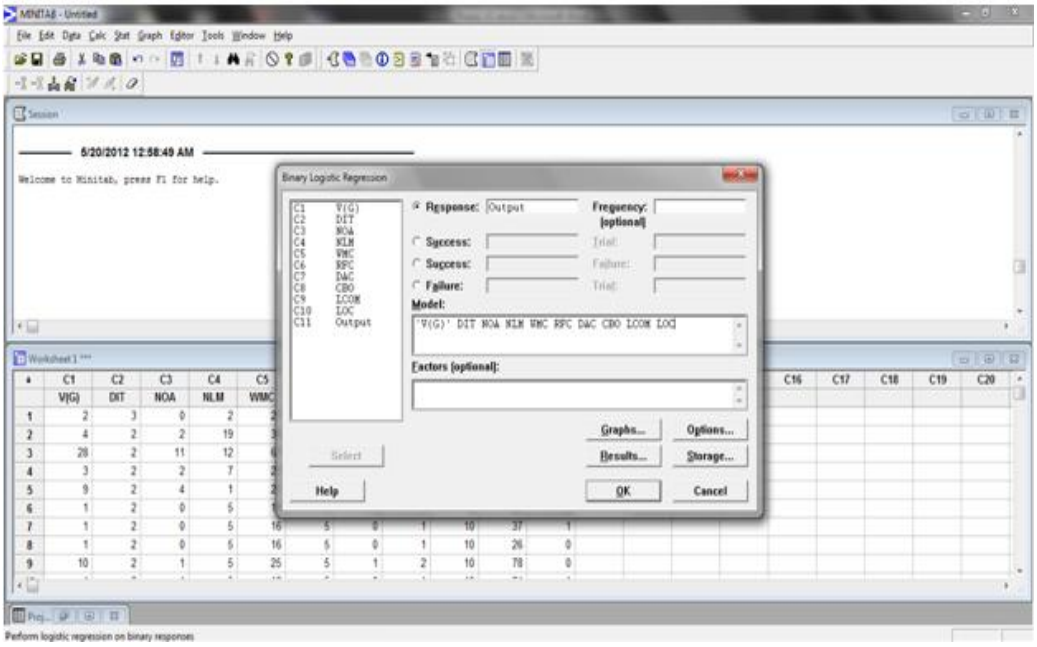


Figure 6-2: Minitab categorical variable screen

6.3 Terminology

In addition to the chapter five terminologies, the following terms are used in the experiment results tables (section):

MLP Linear (L): output layer activation function

MLP Logistic (G): output layer activation function

SVM Linear: kernel function

SVM RBF: kernel function

SVM Sigmoid: kernel function

MC: misclassification percentage

6.4 Experiment Setup

Three setups were selected for the MLP primary phase in regression experiments. The experiment results show good accuracy of the prediction models. Hence, the same setups are selected to be used to build the neural network (MLP) prediction models for classification experiment. In addition, two activation functions for output layer are used in each setup. The neural network (MLP) has two phases, the initial phase to get a first impression of the accuracy of the prediction models and the primary phase where all the selected Java projects are used. Support vector machine (SVM) uses support vector classification (C-SVC) classification methods to build the prediction models using different kernel functions. The target values are converted to either 0 or 1 on all dataset based on the selected threshold value. This section discusses the setup of the classification techniques.

6.4.1 Neural Network (MLP) Initial Phase Setup

For the initial phase, the experiment is conducted on the Eclipse project. Three setups are selected to be part of the classification experiments. Two activation functions of the output layer are used for each setup i.e., linear and logistic. Hold-out sample validation method is used to validate the neural network models as the data is divided to 70% for training and 30% for validation.

6.4.2 Neural Network (MLP) Primary Phase Setup

The same setups that were used in the initial phase are used again in this phase but this time all the selected Java projects are used to create the dataset.

6.4.3 Support Vector Machine (SVM) Setup

Kernel function is one of the important parameters to build an accurate SVM models. Three functions are selected to be used in this experiment i.e., linear, Radial Basis Function (RBF), and sigmoid kernel functions. C-SVC is the classification method that is used in this experiment. The target values are converted to either 0 or 1 on all dataset based on the selected threshold value (0.70. Two validation methods are used i.e., cross-validation with 10 folds to validate models during optimization search process, and hold-out sample with 70% for training and 30% for validation to evaluate and test the final SVM model.

6.4.4 Logistic Regression (LogR) Setup

Logistic regression models are simple to implement and model training is usually much faster when compared to other regression methods such as neural networks. All selected Java projects dataset is used in this experiment. Data is inserted directly to the Minitab worksheet.

6.5 Experiment Results

An experiment is designed to predict class stability based on classification prediction models. Misclassification percentage is used to evaluate the quality of the created models. The minimum of misclassification percentage should not be greater than 25%. Where, the Pearson residual and P-value are used to evaluate the logistic regression model. Table 6-1 describes the estimators. This section shows and discusses the classification experiment results and its observation.

TABLE 6-1: Experiment estimators

Estimator	Description
Misclassification	The percentage of data points or data rows that is misclassified. Misclassification should not be greater than 25%.
Pearson residual	A Measure of how well the observation is predicted by the model. Observations that are poorly fit by the model have high Pearson residuals[65].
P-Value	The probability to reject null hypothesis that no relation between predictor variables and output. A commonly accepted value is 0.05.

6.5.1 Neural Network (MLP) Initial Phase Results

Neural network models are created using the Eclipse project. One project is selected to get a first impression of the accuracy of the neural network models. This section presents the experiment results of neural network (MLP) initial phase for class stability metrics and finally the results and its observation.

6.5.1.1MLP Initial Phase Results for the Stab Metric

Neural network (MLP) models which are created for the Stab metric generate high misclassification parentage even though two activation functions are used for the output layer. Table 6-2 presents the MLP initial phase experiment results for the Stab metric. Only one setup generates a barely acceptable misclassification percentage of 25% under Setup 1.

TABLE 6-2: MLP initial phase experiment results for the Stab metric

Stab – MLP initial phase						
Misclassification %	**Setup 1**		**Setup 2**		**Setup 3**	
	Linear	**Logistic**	**Linear**	**Logistic**	**Linear**	**Logistic**
V1	34.94	28.916	30.12	28.916	26.506	31.325
V2	31.325	25.301	31.325	31.325	30.12	37.349
V3	36.145	34.94	31.325	30.12	30.12	30.12

6.5.1.2 MLP Initial Phase Results for the Grosser Metric

On the other hand, neural network (MLP) models that are created for the Grosser metric generate acceptable results in all experiments. Table 6-3 presents the MLP initial phase experiment results for the Grosser metric. The minimum percentage is 9.639% whereas the maximum is 19.277%.

TABLE 6-3: MLP initial phase experiment results for the Grosser metric

Grosser– MLP initial phase						
Misclassification %	**Setup 1**		**Setup 2**		**Setup 3**	
	Linear	**Logistic**	**Linear**	**Logistic**	**Linear**	**Logistic**
V1	10.843	10.843	12.048	10.843	10.843	9.639
V2	18.072	19.277	13.253	9.639	14.458	9.639
V3	12.048	13.253	12.048	13.253	10.843	14.458

6.5.1.3 MLP Initial Phase Results for the CSM Metric

Like Stab models, neural network (MLP) models for the CSM metric generate unacceptable misclassification percentage in all experiments. Table 6-4 presents the MLP initial phase experiment results for the CSM metric. Most of the experiments exceeded the acceptable misclassification percentage as the generated percentage was above 30%.

TABLE 6-4: MLP initial phase experiment results for the CSM metric

CSM– MLP initial phase						
Misclassification %	**Setup 1**		**Setup 2**		**Setup 3**	
	Linear	**Logistic**	**Linear**	**Logistic**	**Linear**	**Logistic**
V1	33.735	37.349	37.349	37.349	36.145	37.349
V2	37.349	27.711	31.325	34.94	36.145	33.735
V3	33.735	44.578	34.94	34.94	38.554	43.373

6.5.1.4 Observation about MLP Initial Phase Results

- This experiment shows that the neural network (MLP) models for the Grosser metric generate acceptable results in all experiments. Whereas the models for Stab and CSM metrics generate high and unacceptable high misclassification percentage.
- The threshold value of 0.7 is used with Grosser and CSM metrics to convert their output values to either category 0 or 1. Majority of the output for the Eclipse dataset that was created using the Grosser metric was in Category 1,whereas the output for the dataset that was created using the CSM metric was equally distributed among category 0 and 1.
- For the majority of the dataset that was created using the Grosser metric the output was category 1 which represents 92% of dataset. As for the dataset created using the Stab metric, the category 1 output represents72% of the dataset, whereas the output is 57% for CSM metric dataset.. Therefore, increasing the number of any

category in the dataset positively affects the accuracy of the neural network models.

6.5.2 Neural Network (MLP) Primary Phase Results

The selected Java projects are involved in this experiment use the same initial phase setups. This section presents first the experiment results of the neural network (MLP) primary phase for class stability metrics and then the results and its observation.

6.5.2.1 MLP Primary Phase Results for the Stab Metric

This section presents the results of the neural network (MLP) models for the Stab metric, Table 6-5.

TABLE 6-5: MLP primary phase results for the Stab metric

Stab – MLP primary phase												
MC %	**Android project**						**Eclipse project**					
	Setup 1		**Setup 2**		**Setup 3**		**Setup 1**		**Setup 2**		**Setup 3**	
	L	**G**	**L**	**G**	**L**	**G**	**L**	**G**	**L**	**G**	**L**	**G**
V1	10.29	10.29	11.76	10.29	8.824	10.29	34.94	28.91	30.12	28.916	26.50	31.32
V2	19.11	22.05	19.11	19.11	19.11	13.23	31.32	25.30	31.32	31.325	30.12	37.34
V3	19.11	11.76	11.76	14.70	11.76	11.76	36.14	34.94	31.32	30.12	30.12	30.12
MC %	**NetBeans project**						**All-projects**					
	Setup 1		**Setup 2**		**Setup 3**		**Setup 1**		**Setup 2**		**Setup 3**	
	L	**G**	**L**	**G**	**L**	**G**	**L**	**G**	**L**	**G**	**L**	**G**
V1	26.25	27.5	26.25	26.25	28.75	26.25	20.77	21.21	20.34	20.346	22.07	20.34
V2	28.75	30	26.25	22.5	36.25	23.75	26.40	29.00	20.34	20.346	20.34	23.81
V3	33.75	33.75	33.75	32.5	30	32.5	20.34	25.54	20.34	20.346	20.34	20.34

- **Android:** The misclassification results are acceptable for all experiments. The minimum percentage is8.824% whereas the maximum is 22.05% which is still within the acceptable range(i.e. below 25%).
- **Eclipse:** The experiments of this project have already been discussed in the initial phase.
- **NetBeans:** Most of the experiments generate misclassification percentage above the minimum acceptable percentage. Only one setup i.e., Setup 2 Version 2, generates acceptable percentage.
- **All-projects:** Most of the experiments generate acceptable misclassification percentage that is below 25%. Three experiments exceeded 25% value and all of them were part of Setup 1.

6.5.2.2 MLP Primary Phase Results for the Grosser Metric

This section presents the results of the neural network (MLP) models for the Grosser metric, Table 6-6.

TABLE 6-6: MLP primary phase results for the Grosser metric

Grosser – MLP primary phase												
MC %	**Android project**						**Eclipse project**					
	Setup 1		**Setup 2**		**Setup 3**		**Setup 1**		**Setup 2**		**Setup 3**	
	L	**G**	**L**	**G**	**L**	**G**	**L**	**G**	**L**	**G**	**L**	**G**
V1	4.412	4.412	4.412	2.941	4.412	4.412	10.84	10.84	12.04	10.843	10.84	9.639
V2	4.412	4.412	2.941	2.941	4.412	4.412	18.07	19.27	13.25	9.639	14.45	9.639
V3	5.882	5.882	2.941	2.941	8.824	7.353	12.04	13.25	12.04	13.253	10.84	14.45
MC %	**NetBeans project**						**All-projects**					
	Setup 1		**Setup 2**		**Setup 3**		**Setup 1**		**Setup 2**		**Setup 3**	
	L	**G**	**L**	**G**	**L**	**G**	**L**	**G**	**L**	**G**	**L**	**G**
V1	3.75	2.5	3.75	3.75	3.75	3.75	4.762	4.762	4.762	5.195	4.762	5.195
V2	5	3.75	5	5	3.75	5	5.628	7.359	5.628	6.926	5.628	5.195
V3	6.25	5	7.5	6.25	6.25	10	4.762	4.762	7.359	8.225	8.225	4.762

- **Android:** Neural network (MLP) models using this project generate good results in all experiments with a minimum of2.941%.
- **Eclipse:** The experiments of this project have already been discussed in the initial phase.
- **NetBeans:** Like the Android project, the model results are acceptable in all experiments with the minimum value of 2.5%.
- **All-projects:** The experiments in all projects generate good results. The results of all projects put together are also acceptable with a minimum at4.762%**.**

6.5.2.3 MLP Primary Phase Results for the CSM Metric

This section presents the results of the neural network (MLP) models for the CSM metric, Table 6-7.

TABLE 6-7: MLP primary phase results for the CSM metric

CSM– MLP primary phase												
MC %	**Android project**						**Eclipse project**					
	Setup 1		**Setup 2**		**Setup 3**		**Setup 1**		**Setup 2**		**Setup 3**	
	L	**G**	**L**	**G**	**L**	**G**	**L**	**G**	**L**	**G**	**L**	**G**
V1	2.941	1.471	2.941	2.941	1.471	2.941	33.73	37.34	37.34	37.349	36.14	37.34
V2	4.412	4.412	4.412	1.471	5.882	4.412	37.34	27.71	31.32	34.94	36.14	33.73
V3	4.412	2.941	4.412	5.882	7.353	4.412	33.73	44.57	34.94	34.94	38.55	43.37
MC %	**NetBeans project**						**All-projects**					
	Setup 1		**Setup 2**		**Setup 3**		**Setup 1**		**Setup 2**		**Setup 3**	
	L	**G**	**L**	**G**	**L**	**G**	**L**	**G**	**L**	**G**	**L**	**G**
V1	20	16.25	20	17.5	17.5	18.75	23.81	24.67	21.64	23.377	22.94	23.37
V2	21.25	25	21.25	21.25	26.25	23.75	34.19	31.60	31.16	31.169	26.84	28.57
V3	22.5	17.5	18.75	21.25	26.25	25	24.67	25.10	26.40	29.004	25.10	25.10

- **Android:** All the experiment results are acceptable and the minimum value is1.471% which is the lowest percentage when compared to other stability metrics in the same project.
- **Eclipse:** The experiments of this project have already been discussed in the initial phase.
- **NetBeans:** The experiment results are acceptable with only two experiment runs, both in Setup 3generated unacceptable misclassification percentages that are above 25%.
- **All-projects:** The experiment results are mixed, with some experiments generating acceptable percentage whereas others exceed the 25% value. The minimum percentage of the misclassification in the All-project is21.64%.

6.5.2.4 Observation about MLP Primary Phase Results

- The prediction model for the CSM metric generated the lowest misclassification percentage that was obtained from the Android project.
- Once again, the majority of the category 1 output affects the model accuracy positively. For instance, the Android project dataset that was created using CSM metric output was 96% category 1. Therefore, the model for CSM metric generates the lowest misclassification percentage. While the NetBeans and the Eclipse datasets have 75% and 57% of category 1 output respectively for which the accuracy of the models was decreased. Thus, when dataset consists of almost one

category either 0 or 1, the accuracy of the neural network is more with lower misclassification percentage.

6.5.3 Support Vector Machine (SVM) Results

Three kernel functions are used in these experiments and misclassification percentage is used to measure the quality and accuracy of support vector machine (SVM) models for class stability metrics. This section presents the results of SVM models and its observations.

6.5.3.1 SVM Results for the Stab Metric

This section presents the results of support vector machine (SVM) models for the Stab metric, Table 6-8.

TABLE 6-8: SVM results for the Stab Metric

Stab - SVM									
MC %	**Android project**			**Eclipse project**			**NetBeans project**		
	Linear	**RBF**	**Sigmoid**	**Linear**	**RBF**	**Sigmoid**	**Linear**	**RBF**	**Sigmoid**
V1	10.294	8.824	10.294	28.916	27.711	28.916	22.5	22.5	22.5
V2	10.294	14.706	19.118	26.506	26.506	24.096	22.5	22.5	22.5
V3	10.294	10.294	10.294	26.506	30.12	28.916	22.5	22.5	25
V4	10.294	10.294	8.824	26.506	26.506	24.096	22.5	22.5	27.5

Stab (All-projects) - SVM			
MC %	**Linear**	**RBF**	**Sigmoid**
V1	20.346	20.346	20.779
V2	20.346	21.212	20.346
V3	20.346	20.346	20.346

- **Android:** The misclassification percentage is acceptable in all the experiments using different kernel functions. The minimum percentageis8.824% where the maximum is 19.118%.
- **Eclipse:** Unacceptable percentage in all experiments.
- **NetBeans:** All the experiments generate acceptable percentage around22.5% except one experiment run that used sigmoid function.
- **All-projects:** Collecting all projects together in one dataset and building support vector machine (SVM) models generate good and acceptable results in all experiments using different activation functions. The minimum percentage is 20.346%.

6.5.3.2 SVM Results for the Grosser Metric

This section presents the results of support vector machine (SVM) models for the Grosser metric, Table 6-9.

TABLE 6-9: SVM results for the Grosser Metric

Grosser- SVM									
MC %	**Android project**			**Eclipse project**			**NetBeans project**		
	Linear	**RBF**	**Sigmoid**	**Linear**	**RBF**	**Sigmoid**	**Linear**	**RBF**	**Sigmoid**
V1	2.941	2.941	2.941	8.434	8.434	8.434	3.75	3.75	3.75
V2	2.941	2.941	2.941	8.434	8.434	8.434	3.75	3.75	3.75
V3	2.941	2.941	2.941	8.434	8.434	8.434	3.75	3.75	3.75
V4	2.941	2.941	2.941	8.434	8.434	8.434	3.75	3.75	3.75

Grosser(All-projects) - SVM			
MC %	**Linear**	**RBF**	**Sigmoid**
V1	4.762	4.762	4.762
V2	4.762	4.762	4.762
V3	4.762	4.762	4.762

- **Android:** All the experiments generate the same acceptable results i.e.2.941%.
- **Eclipse:** The misclassification percentage is higher than that of the Android project but still it is within the acceptable rang and is constant for all experiments. The percentage is8.434%**.**
- **NetBeans:** Like the pervious projects, the models generate acceptable and consistent misclassification percentage.
- **All-projects:** All the experiments generate the same misclassification percentage which can be considered as acceptable percentage.

6.5.3.3 SVM Results for the CSM Metric

This section presents the results of support vector machine models for the CSM metric, Table 6-10.

TABLE 6-10: SVM results for the CSM Metric

CSM- SVM									
MC %	**Android project**			**Eclipse project**			**NetBeans project**		
	Linear	**RBF**	**Sigmoid**	**Linear**	**RBF**	**Sigmoid**	**Linear**	**RBF**	**Sigmoid**
V1	4.412	4.412	4.412	39.759	43.373	38.554	26.25	18.75	26.25
V2	4.412	4.412	4.412	34.94	40.964	36.145	26.25	23.75	26.25
V3	4.412	4.412	4.412	39.759	38.554	40.964	26.25	12.5	33.75
V4	4.412	5.882	4.412	38.554	40.964	39.759	26.25	21.25	26.25

CSM(All-projects) - SVM			
MC %	**Linear**	**RBF**	**Sigmoid**
V1	25.974	22.944	25.974
V2	25.541	28.139	25.541
V3	25.541	24.242	24.242

- **Android:** All experiments generate good and acceptable results. The minimum percentage is4.412%
- **Eclipse:** Unacceptable misclassification percentage in all experiments.
- **NetBeans:** Experiments done using linear and sigmoid as a kernel functions generate unacceptable percentage. The confusion matrix shows that the weight of the correctly classified cases for category 0 was low in both functions. The RBF function generates acceptable percentage. The minimum percentage is12.5% and the weight of the correctly classified cases for both categories were acceptable.
- **All-projects:** Experiments done using linear function generate percentage above 25% as the weight of the correctly classified cases for category 0 was low, whereas with other functions the results vary as some lie within the acceptable percentage while some cross the acceptable percentage 25% and this time the percentage of misclassified cases was high.

6.5.3.4 Observation about SVM Results

Some key observations concerning the SVM results are summarized below:

- The models for the Grosser metric generate the lowest misclassification percentage in all the selected Java projects. CSM models perform better than Stab in Android project whereas Stab models outperform CSM models in other projects.
- The models for the Grosser metric generate the same misclassification percentage value in each project as the classification method used the same parameter values.

Stab and CSM models generate the same misclassification percentage value in the NetBeans and Android projects respectively except for some experiment runs where the method parameter values are changed.

- The models for the CSM metric using RBF kernel function generate good results in NetBeans and All-projects compared to the other functions used. The confusion matrix shows that the percentage of the correctly classified cases was high compared to the other projects.
- Similar to the result observation of the neural network experiments, the majority of category 1output in a dataset affects the accuracy of the classification model positively.
- The selected Java projects are well defined projects. The methods are rarely deleted and they are added to the new version. Grosser metric measures class stability based on only one factor, Number of Methods (NOM). As a result, almost all the classes are absolutely stable in all selected Java projects, which means that the category 1 forms the majority output for the dataset of Java projects. Therefore, Grosser models generate the better misclassification percentage compared to Stab and CSM models.

6.5.4 Logistic Regression Results

Logistic regression is used to build classification models for software quality. Therefore, an experiment is designed and performed using logistic regression on class stability metrics to observe the results.

6.5.4.1 LogR Results for the Stab metric

The Pearson residuals percentages are high in the all selected Java projects which indicate that model fit poorly. Table 6-11 presents the LogR results for the Stab metric. Although, the P-value of LogR model in NetBeans is less than the common value "0.05", but still the Pearson residuals percentage is not acceptable.

TABLE 6-11: LogR results for the Stab metric

Stab metric		
Project	**Pearson residuals**	**P-Value**
Android	89.95%	0.393
Eclipse	196.16%	0.884
NetBeans	169.47%	0.031
All-projects	117.03%	0.891

6.5.4.2 LogR Results for the Grosser metric

The LogR models for the Grosser metric also generate high Pearson residuals percentages and unacceptable P-values except for the All-projects dataset. Table 6-12 presents the LogR results for the Grosser metric.

TABLE 6-12: LogR results for the Grosser metric

Grosser metric		
Project	**Pearson residuals**	**P-Value**
Android	99.63%	0.167
Eclipse	207.25%	0.738
NetBeans	117.03%	0.891
All-projects	562.41%	0.001

6.5.4.3 LogR Results for the CSM metric

The P-values were less than the common value in the all selected Java projects. However, the logistic regression model for the CSM metric also shows high Pearson residuals percentages like other stability metrics. Table 6-13 presents the LogR results for the CSM metric.

TABLE 6-13: LogR results for the CSM metric

CSM metric		
Project	**Pearson residuals**	**P-Value**
Android	149.73%	0
Eclipse	264.94%	0.023
NetBeans	226.80%	0
All-projects	688.59%	0

6.5.4.4 Observation about LogR Results

- The Pearson residuals percentages are high for all the stability metrics. As a result, the predictor variables do not accurately predicts the target variable using logistic regression.
- Some experiments generate acceptable P-value. However, the Pearson residuals percentages of these experiments are very high and unacceptable.
- Logistic regression builds model based on statistical relationship between software metrics and quality factor. The low percentage of the Pearson residuals indicates good fit of the model. However, the results show high percentage of the Pearson residuals. Hence, logistic regression may not be a good choice to predict stability using software metrics as measured by class stability metrics.

6.6 Discussion

Classification experiments were conducted using different techniques to build models for class stability prediction using the selected software metrics. The pervious sections presented the results of different experiments and focused primarily on the minimum misclassification percentage. To evaluate the accuracy of the created prediction models, the average and standard deviation values are also calculated. These values can be used as an indicator of the quality and accuracy of the prediction models. This section presents and discusses the average and standard deviation results for the neural network (MLP) and support vector machine (SVM).

6.6.1 Neural Network (MLP)

The primary phase is used to calculate the average and standard deviation values as all the selected Java projects were used. Table 6-11 shows the average and standard deviation results of neural network primary phase.

TABLE 6-14: MLP Avg & Std results

Neural Network - MLP						
Stability Metrics	**Average**			**Standard Deviation**		
	Android	**Eclipse**	**NetBeans**	**Android**	**Eclipse**	**NetBeans**
Stab	14.134	31.124	29.166	4.224	3.099	3.882
Grosser	4.5752	12.516	5	1.5862	2.714	1.767
CSM	3.8399	36.144	21.111	1.606	3.854	3.147

- **Stab**: Neural network (MLP) models for the Stab metric show good performance for Android project. As a result, the average and the standard deviation values of these models are acceptable. On the other hand, the models for the Stab metric generate unacceptable misclassification percentage in the most of the experiments of the Eclipse and NetBeans projects. Therefore, the average values are above the acceptable percentage of 25%.
- **Grosser**: Unlike models for the Stab metric, the Grosser models perform well in all selected Java projects. Consequently, the average and standard deviation values for selected Java projects are all acceptable.
- **CSM**: The CSM models also generate good misclassification percentages in all the experiments of the Android project, so the average and standard deviation are acceptable. While the CSM models generate unacceptable percentage in all the experiments of the Eclipse project. As a result the average value in Eclipse project is unacceptable. Finally, the CSM models' average and standard deviation in the

NetBeans project are acceptable and under the 25% range. Figure6-2 illustrates the neural network models performance.

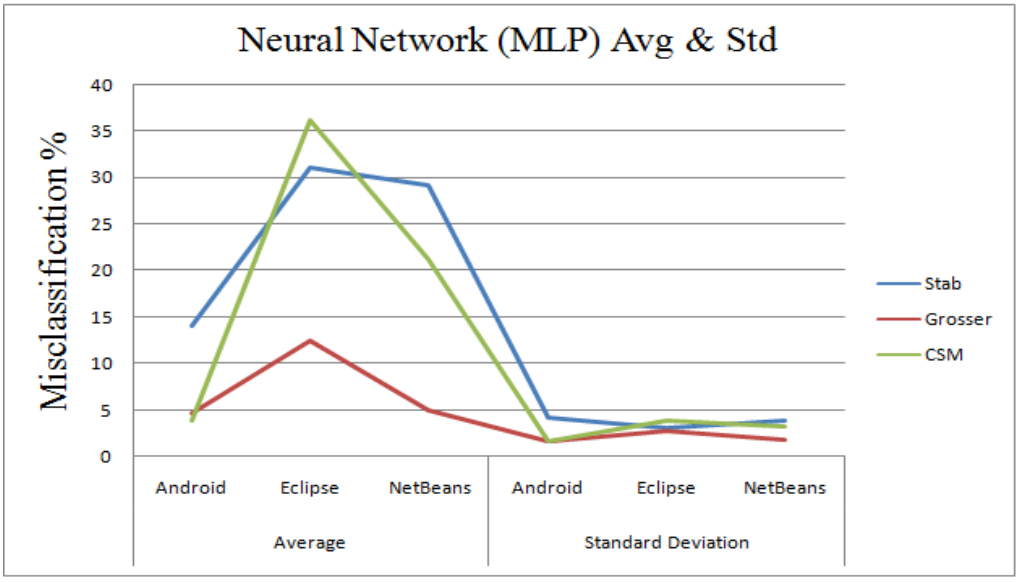


Figure 6-3: MLP models performance

6.6.2 Support Vector Machine (SVM)

The average and standard deviation are calculated for all SVM experiments in the selected Java projects. Table 6-12 shows the average and standard deviation results of support vector machine (SVM).

TABLE 6-15: SVM Avg & Std results

Support Vector Machine (SVM)						
Stability Metrics	**Average**			**Standard Deviation**		
	Android	**Eclipse**	**NetBeans**	**Android**	**Eclipse**	**NetBeans**
Stab	11.152	27.108	23.125	2.9048	1.8877	1.5539
Grosser	2.941	8.434	3.75	4.64E-16	0	0
CSM	4.5345	39.357	24.479	0.42435	2.2587	5.1802

- **Stab**: The prediction models for the Stab metric generate acceptable average and standard deviation values in the Android and NetBeans projects. Whereas the models in the Eclipse show unacceptable average value.
- **Grosser**: Once again, the SVM models for the Grosser metric show good average and standard deviation values in all projects.
- **CSM**: The average and standard deviation values are acceptable for all models in the Android project. However, the models for the Eclipse project show unacceptable average value. While the models for NetBeans show acceptable average value but the standard deviation makes the average upper bound exceed the 25% mark. Figure6-3 illustrates the neural network models performance.

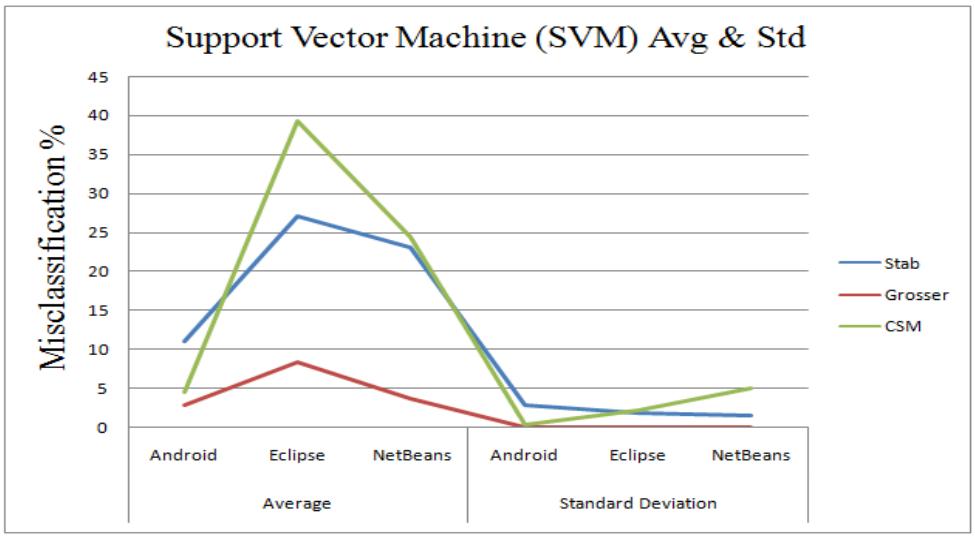


Figure 6-4: SVM models performance

6.6.3 Logistic Regression (LogR)

The experiment results of all Java projects show that the logistic regression is not able to predict class stability using the selected software metrics.

6.6.4 Overall

This section presents and discusses the average and standard deviation results of neural network (MLP) and support vector machine (SVM) models in All-projects dataset. The average and standard deviation values are present in Table 6-13.

TABLE 6-16: Overall Avg & Std results

Stability Metrics	Neural Network –MLP All-projects		Support Vector Machine – SVM All-projects	
	Average	Standard Deviation	Average	Standard Deviation
Stab	21.81317	2.610779	20.49033	0.306177
Grosser	5.772056	1.250999	4.762	0
CSM	26.59933	3.554968	25.34867	1.453911

- The models for Grosser and Stab metrics show acceptable average and standard deviation values in the neural network (MLP) and the support vector machine (SVM), where model for the CSM metric show unacceptable values for both techniques.
- The best average and standard deviation values were obtained from Grosser metric models compared to all other metrics.
- The models for the Grosser metric generate the same misclassification percentage value in each project. As a result, the standard deviation value is 0.
- Once again, the majority of category 1 output in the dataset is the key to improving the accuracy of the classification models of neural network and support vector machine. The dataset that is used for the Grosser metric consists of 95% of category 1. Whereas datasets for Stab and CSM metrics consists of 80% and 72%

category 1 outputs. Therefore, models for Grosser metric generate the best misclassification results.

- Grosser and Stab metrics accept the hypotheses, where the CSM metric rejects it. Hypothesis: Software metrics are able to predict class stability as measured by class stability metrics.
- As the classification models' accuracy depend on the majority of any category on the dataset. Hence, the classification method may not be a good choice to predict the stability using software metrics. Figure 6-4 illustrates the models performance in All-projects.

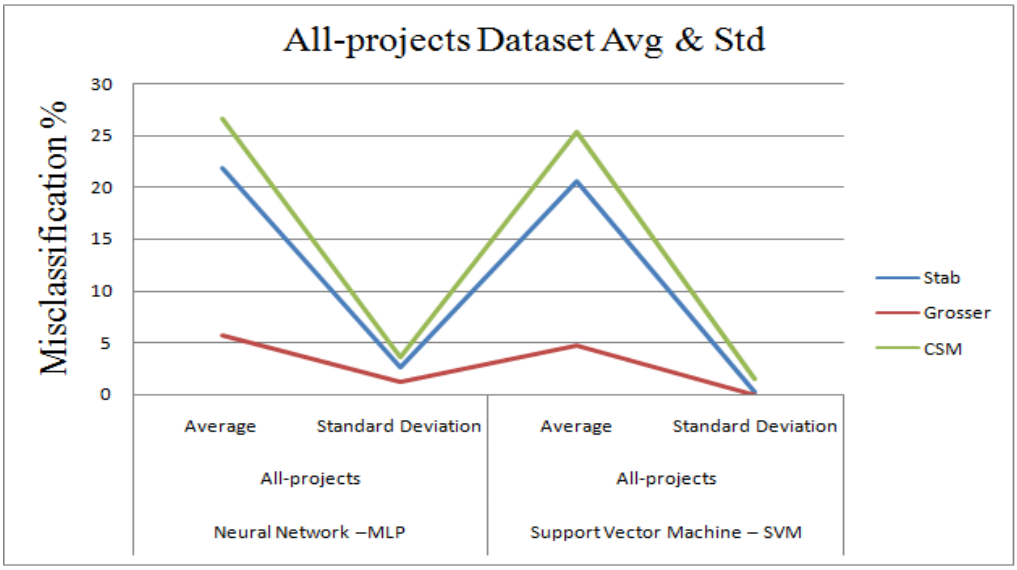


Figure 6-5: Overall models performance

CHAPTER 7

CONCLUSION & FUTURE WORK

7.1 Conclusion

The goal of this research was to predict the class stability using software metrics, therefore a prediction model was created and its performance was evaluated. A literature review of various AI techniques which are used to predict the software quality was conducted. The review shows that there are many AI techniques that are proposed by researchers to predict different factors of software quality such as maintainability, reliability using software metrics as predictor variables.

Building prediction models for class stability includes experimentation and hypothesis testing. Therefore, a set of hypotheses were proposed stating that the software metrics are able to predict class stability as measured by the class stability metrics. The

experiments' data had an important role in evaluating the model's accuracy. The data was obtained from three open source Java projects i.e., Android, Eclipse, and NetBeans. Many software metrics were selected which belonged to different metric categories such as coupling, cohesion, and complexity. These metrics were used as predictor variables in the prediction model. Class stability metrics were used to measure the stability of the class in different versions for all selected Java projects in order to use the stability output as a target variable for the prediction model. The proposed hypotheses were tested on two different experiments i.e., regression and classification. Multilayer perceptron neural network and support vector machine techniques were used in both the experiments. In addition to the previous techniques, Multi linear regression technique was also used in the regression experiment and logistic regression was used for classification experiment.

The regression experiment results showed that the prediction model of the CSM metric is accurate. The performance of Grosser metric model was low in some of the selected projects, but the overall the performance was good. This is due to the fact that the CSM metric uses many factors to measure class stability whereas the Grosser metric is based only on one factor. Hence, the CSM metric reflects the selected software metrics better than the Grosser metrics. The model of CII metric showed that this metric cannot be used to predict class stability. The selected software metrics were able to predict class stability as measured by CSM and Grosser class stability metrics which means that they accept the proposed hypotheses, whereas the selected software metrics were not able to predict class stability as measured by the CII metric which means that it rejects the proposed hypothesis.

The classification experiment results showed the accuracy of the prediction model of Grosser and Stab metrics. The CSM metric model showed low performance and unacceptable misclassification error percentage. However, the classification model's accuracy depends on the majority of any category either 0 or 1 on the dataset. Hence, the classification method may not be a good choice to predict the class stability that is measured by the class stability metrics.

In addition, the multi linear regression model that was used in regression experiment and the logistic regression model that used in classification experiment show unacceptable results in all experiments. Therefore, these techniques may not be a good choice to predict the class stability that is measured by the class stability metrics.

7.2 Threats to Validity

A list of threats to validity of this research experiments is presented and discussed in this section. These threats may affect the research's conclusion.

1. The process of calculating the class stability metrics' values is semi-automated. It involves manual process during the class stability measurement activities. Manual processes are prone to human errors which may affect the measurement accuracy.
2. The CSM metric is theoretically validated. However, for the other class stability metrics, we assumed that they are theoretically valid. Hence, we did not validate these metrics theoretically.

3. The experiment data has an important role in the model's accuracy. The research conclusions were based on the selected Java projects that were used to build the prediction models.

7.3 Future Work

Further investigations and researches can be done in this area. Future work in this area of research can include the following:

❖ **Investigate the inconsistencies in the measurement of the class stability metrics**

The proposed metrics use different quality factors to measure the class stability. As a result, the measurements won't be identical using these metrics. Therefore, relying on the measurements of only one metric may not lead to the same decision when considering other metrics, which can cause confusion when trying to come up with the best decision. Moreover, which metrics of class stability best measures stability? Therefore an investigation is required to develop a framework that assists to correlate the probability distribution of error to the measurements.

❖ **Predicting class stability using different artificial intelligent techniques.**

Neural network – Multi Layer Perceptron (MLP) and Support Vector Machine (SVM) were used to predict the object-oriented class stability. Other possibilities for predicting stability can be conducted using other artificial intelligent techniques such as decision tree, fuzzy logic, Bayesian network, etc.

❖ **Investigate the relationship between stability and other quality factors.**

The quality factors such as stability, maintainability, reliability, efficiency, etc. are the key factors in assessing the quality of the software. A research study can be conducted to investigate and find out whether any correlation exists between these factors and if so, how they affect each other.

❖ **Predicting class stability using different dataset.**

The experiment data has an important role in evaluating the model's accuracy. The data used was obtained from three open source Java projects. These projects are well-defined projects in which features or methods are usually deprecated rather than immediately removed. Further study on different datasets can be done to validate the accuracy of the prediction models.

Appendix A

Class Stability Metrics Factors

TABLE A- 1 Android Class Attributes V1.5

Android Class Attributes			
Class Name	**Version 1.5**		
	P-NOM	**LOC**	**NOM**
Typeface	8	66	14
Scroller	20	195	21
DeleteEventHelper	4	190	8
OnScreenHint	16	157	20
EmailAddressValidator	2	13	2
MessagingListener	28	84	28
AlarmManager	5	53	6
NotificationManager	3	62	5
StatusBarManager	7	69	8
AppWidgetProvider	6	44	6
ActivityNotFoundException	2	11	2
ContentUris	3	14	3
MutableContextWrapper	2	9	2
ReceiverCallNotAllowedException	1	7	1
SyncContext	5	33	5
TypedArray	28	372	31
CharArrayBuffer	2	11	2
CursorIndexOutOfBoundsException	2	9	2
DataSetObserver	2	7	2
SQLException	2	9	2
StaleDataException	2	12	2
SQLiteAbortException	2	7	2
SQLiteClosable	5	32	5
SQLiteConstraintException	2	7	2
SQLiteDatabaseCorruptException	2	7	2
SQLiteDiskIOException	2	7	2
SQLiteDoneException	2	7	2
SQLiteException	2	8	2
SQLiteFullException	2	7	2
SQLiteMisuseException	2	7	2
SQLiteOpenHelper	7	110	7
SQLiteStatement	4	94	8
GpsSatellite	7	44	9
LocationProvider	11	29	12

Android Class Attributes			
Class Name	**Version 1.5**		
	P-NOM	**LOC**	**NOM**
Ringtone	6	149	13
FocusFinderHelper	7	26	7
Gravity	5	159	5
InflateException	4	15	4
SoundEffectConstants	1	25	2
TouchDelegate	2	64	2
WindowManagerImpl	12	249	17
AccelerateInterpolator	4	25	4
BaseInputConnection	23	390	27
DocumentBuilder	14	57	14
FactoryConfigurationError	6	31	6
ParserConfigurationException	2	9	2
SAXParser	19	129	19
AbstractMethodError	2	10	2
ArrayStoreException	2	10	2
AssertionError	8	30	8
ClassFormatError	2	10	2
Error	4	16	4
Exception	4	16	4
IllegalAccessError	2	10	2
IllegalAccessException	2	10	2
IllegalStateException	4	16	4
Math	44	157	47
Number	7	16	7
OutOfMemoryError	2	10	2
Process	6	11	6
RuntimeException	4	16	4
RuntimePermission	2	41	2
SecurityException	4	16	4
StackOverflowError	2	10	2
StrictMath	44	175	47
StringIndexOutOfBoundsException	3	14	3
ThreadDeath	1	6	1
TypeNotPresentException	2	12	2

Android Class Attributes			
Class Name	**Version 1.5**		
	P-NOM	**LOC**	**NOM**
UnknownError	2	10	2
VerifyError	2	10	2
AlgorithmParameterGenerator	11	82	11
AuthProvider	4	13	4
CollationElementIterator	11	43	12
Environment	5	40	6
SystemProperties	6	66	9
Vibrator	4	35	4

TABLE A- 2 Android Class Attributes V1.6

Android Class Attributes			
Class Name	**Version 1.6**		
	P-NOM	**LOC**	**NOM**
Typeface	10	73	17
Scroller	20	195	21
DeleteEventHelper	4	190	8
OnScreenHint	9	126	13
EmailAddressValidator	2	13	2
MessagingListener	34	96	34
AlarmManager	5	53	6
NotificationManager	3	62	5
StatusBarManager	7	69	8
AppWidgetProvider	6	42	6
ActivityNotFoundException	2	11	2
ContentUris	3	14	3
MutableContextWrapper	2	9	2
ReceiverCallNotAllowedException	1	7	1
SyncContext	5	33	5
TypedArray	28	373	31
CharArrayBuffer	2	11	2

Android Class Attributes			
Class Name	**Version 1.6**		
	P-NOM	**LOC**	**NOM**
CursorIndexOutOfBoundsException	2	9	2
DataSetObserver	2	7	2
SQLException	2	9	2
StaleDataException	2	12	2
SQLiteAbortException	2	7	2
SQLiteClosable	5	32	5
SQLiteConstraintException	2	7	2
SQLiteDatabaseCorruptException	2	7	2
SQLiteDiskIOException	2	7	2
SQLiteDoneException	2	7	2
SQLiteException	2	8	2
SQLiteFullException	2	7	2
SQLiteMisuseException	2	7	2
SQLiteOpenHelper	7	110	7
SQLiteStatement	4	94	8
GpsSatellite	7	44	9
LocationProvider	12	49	12
Ringtone	6	149	13
FocusFinderHelper	7	26	7
Gravity	5	159	5
InflateException	4	15	4
SoundEffectConstants	1	25	2
TouchDelegate	2	64	2
WindowManagerImpl	12	249	17
AccelerateInterpolator	4	30	4
BaseInputConnection	23	402	27
DocumentBuilder	14	57	14
FactoryConfigurationError	6	31	6
ParserConfigurationException	2	9	2
SAXParser	17	129	19
AbstractMethodError	2	10	2
ArrayStoreException	2	10	2
AssertionError	8	30	8
ClassFormatError	2	10	2

Android Class Attributes			
Class Name	**Version 1.6**		
	P-NOM	**LOC**	**NOM**
Error	4	16	4
Exception	4	16	4
IllegalAccessError	2	10	2
IllegalAccessException	2	10	2
IllegalStateException	4	16	4
Math	44	157	47
Number	7	16	7
OutOfMemoryError	2	10	2
Process	6	11	6
RuntimeException	4	16	4
RuntimePermission	2	41	2
SecurityException	4	16	4
StackOverflowError	2	10	2
StrictMath	44	175	47
StringIndexOutOfBoundsException	3	14	3
ThreadDeath	1	6	1
TypeNotPresentException	2	12	2
UnknownError	2	10	2
VerifyError	2	10	2
AlgorithmParameterGenerator	11	82	11
AuthProvider	4	13	4
CollationElementIterator	11	43	12
Environment	5	40	6
SystemProperties	6	66	9
Vibrator	4	35	4

TABLE A- 3 Android Class Attributes V2.1

Android Class Attributes			
Class Name	**Version 2.1**		
	P-NOM	**LOC**	**NOM**
Typeface	11	74	18
Scroller	21	199	22
DeleteEventHelper	4	189	8
OnScreenHint	5	112	9
EmailAddressValidator	2	13	2
MessagingListener	19	58	19
AlarmManager	5	53	6
NotificationManager	6	70	7
StatusBarManager	7	69	8
AppWidgetProvider	6	42	6
ActivityNotFoundException	2	11	2
ContentUris	3	14	3
MutableContextWrapper	2	9	2
ReceiverCallNotAllowedException	1	7	1
SyncContext	4	34	5
TypedArray	28	373	31
CharArrayBuffer	2	11	2
CursorIndexOutOfBoundsException	2	9	2
DataSetObserver	2	7	2
SQLException	2	9	2
StaleDataException	2	12	2
SQLiteAbortException	2	7	2
SQLiteClosable	5	32	5
SQLiteConstraintException	2	7	2
SQLiteDatabaseCorruptException	2	7	2
SQLiteDiskIOException	2	7	2
SQLiteDoneException	2	7	2
SQLiteException	2	8	2
SQLiteFullException	2	7	2
SQLiteMisuseException	2	7	2
SQLiteOpenHelper	7	110	7
SQLiteStatement	4	94	8
GpsSatellite	7	44	9
LocationProvider	12	49	12

Android Class Attributes			
Class Name	**Version 2.1**		
	P-NOM	**LOC**	**NOM**
Ringtone	6	149	13
FocusFinderHelper	7	26	7
Gravity	5	159	5
InflateException	4	15	4
SoundEffectConstants	1	25	2
TouchDelegate	2	64	2
WindowManagerImpl	12	249	17
AccelerateInterpolator	4	30	4
BaseInputConnection	23	402	27
DocumentBuilder	14	57	14
FactoryConfigurationError	6	31	6
ParserConfigurationException	2	9	2
SAXParser	17	129	19
AbstractMethodError	2	10	2
ArrayStoreException	2	10	2
AssertionError	8	30	8
ClassFormatError	2	10	2
Error	4	16	4
Exception	4	16	4
IllegalAccessError	2	10	2
IllegalAccessException	2	10	2
IllegalStateException	4	16	4
Math	44	171	47
Number	7	16	7
OutOfMemoryError	2	10	2
Process	6	11	6
RuntimeException	4	16	4
RuntimePermission	2	41	2
SecurityException	4	16	4
StackOverflowError	2	10	2
StrictMath	44	175	47
StringIndexOutOfBoundsException	3	14	3
ThreadDeath	1	6	1
TypeNotPresentException	2	12	2

Android Class Attributes			
Class Name	**Version 2.1**		
	P-NOM	**LOC**	**NOM**
UnknownError	2	10	2
VerifyError	2	10	2
AlgorithmParameterGenerator	11	82	11
AuthProvider	4	13	4
CollationElementIterator	11	43	12
Environment	5	40	6
SystemProperties	6	52	12
Vibrator	4	36	4

TABLE A- 4 Android Class Attributes V2.3.1

Android Class Attributes			
Class Name	**Version 2.3.1**		
	P-NOM	**LOC**	**NOM**
Typeface	11	77	18
Scroller	21	198	22
DeleteEventHelper	4	190	8
OnScreenHint	5	111	9
EmailAddressValidator	2	11	2
MessagingListener	19	56	19
AlarmManager	6	59	7
NotificationManager	6	71	7
StatusBarManager	6	64	7
AppWidgetProvider	6	42	6
ActivityNotFoundException	2	11	2
ContentUris	3	14	3
MutableContextWrapper	2	9	2
ReceiverCallNotAllowedException	1	7	1
SyncContext	4	38	5
TypedArray	28	393	32
CharArrayBuffer	2	11	2

Android Class Attributes			
Class Name	Version 2.3.1		
	P-NOM	LOC	NOM
CursorIndexOutOfBoundsException	2	9	2
DataSetObserver	2	7	2
SQLException	2	9	2
StaleDataException	2	12	2
SQLiteAbortException	2	7	2
SQLiteClosable	5	51	6
SQLiteConstraintException	2	7	2
SQLiteDatabaseCorruptException	2	7	2
SQLiteDiskIOException	2	7	2
SQLiteDoneException	2	7	2
SQLiteException	2	8	2
SQLiteFullException	2	7	2
SQLiteMisuseException	2	7	2
SQLiteOpenHelper	7	114	7
SQLiteStatement	4	79	8
GpsSatellite	7	44	9
LocationProvider	12	40	12
Ringtone	6	159	13
FocusFinderHelper	7	26	7
Gravity	5	159	5
InflateException	4	15	4
SoundEffectConstants	1	25	2
TouchDelegate	2	64	2
WindowManagerImpl	12	249	17
AccelerateInterpolator	4	30	4
BaseInputConnection	25	458	31
DocumentBuilder	13	85	15
FactoryConfigurationError	6	30	6
ParserConfigurationException	2	9	2
SAXParser	20	159	20
AbstractMethodError	2	10	2
ArrayStoreException	2	10	2
AssertionError	8	30	8
ClassFormatError	2	10	2

Android Class Attributes			
Class Name	**Version 2.3.1**		
	P-NOM	**LOC**	**NOM**
Error	4	16	4
Exception	4	16	4
IllegalAccessError	2	10	2
IllegalAccessException	2	10	2
IllegalStateException	4	16	4
Math	54	405	59
Number	7	16	7
OutOfMemoryError	2	10	2
Process	6	11	6
RuntimeException	4	16	4
RuntimePermission	2	41	2
SecurityException	4	16	4
StackOverflowError	2	10	2
StrictMath	54	283	59
StringIndexOutOfBoundsException	3	13	3
ThreadDeath	1	6	1
TypeNotPresentException	2	12	2
UnknownError	2	10	2
VerifyError	2	10	2
AlgorithmParameterGenerator	11	80	11
AuthProvider	4	13	7
CollationElementIterator	11	43	12
Environment	15	110	16
SystemProperties	6	52	12
Vibrator	4	52	4

TABLE A- 5 Eclipse Class Attributes V2.0

Eclipse Class Attributes			
Class Name	**Version 2.0**		
	P-NOM	**LOC**	**NOM**
JDTCompilerAdapter	2	108	3
ClasspathDirectory	4	93	8
FileFinder	2	36	2
CodeFormatter	15	1872	50
AbstractMethodDeclaration	22	297	24
AbstractVariableDeclaration	3	24	3
Argument	5	62	5
ArrayAllocationExpression	6	126	6
ArrayInitializer	6	154	6
ClassFileReader	32	543	38
ClassFileStruct	16	161	16
ClassFormatException	3	44	3
FieldInfo	13	233	16
InnerClassInfo	7	89	8
MethodInfo	13	161	16
ConditionalFlowInfo	19	82	20
ExceptionHandlingFlowContext	6	142	7
FlowInfo	23	48	23
InitializationFlowContext	3	64	3
LabelFlowContext	3	35	4
LoopingFlowContext	7	121	9
BooleanConstant	5	22	5
ByteConstant	11	40	11
CompilerOptions	7	522	7
Constant	42	1254	42
IntConstant	11	41	11
LongConstant	11	40	11
StringConstant	5	21	5
RecoveredBlock	17	203	18
RecoveredField	12	117	12
RecoveredImport	7	31	7
RecoveredInitializer	12	172	12
RecoveredMethod	16	316	16
BufferedContent	9	53	9

Eclipse Class Attributes			
Class Name	**Version 2.0**		
	P-NOM	**LOC**	**NOM**
CompareConfiguration	26	174	27
CompareUI	12	66	13
CompareViewerPane	5	57	6
CompareViewerSwitchingPane	20	200	23
HistoryItem	6	30	6
NavigationAction	4	27	4
ResourceNode	14	92	14
AddFromHistoryAction	3	100	5
BinaryCompareViewer	4	94	5
ChangePropertyAction	4	31	4
CompareAction	2	24	2
CompareMessages	1	16	2
DocLineComparator	6	111	8
DocumentManager	4	42	4
ExceptionHandler	7	62	8
ImageMergeViewer	7	101	8
ResizableDialog	4	112	5
SimpleTextViewer	3	44	5
DiffContainer	7	51	7
DiffElement	7	32	7
DiffNode	23	175	24
DocumentRangeNode	17	175	19
DebugEvent	7	114	7
DebugException	1	13	1
Launch	23	194	25
Context	9	45	9
HrefUtil	4	54	4
Link	4	25	4
Toc	11	110	11
TocFile	13	74	13
TocFileParser	7	92	7
TocManager	5	136	9
URLCoder	2	49	4
OverlayIcon	8	95	8

Eclipse Class Attributes			
Class Name	**Version 2.0**		
	P-NOM	**LOC**	**NOM**
PDERuntimePlugin	10	67	12
PDERuntimePluginImages	2	107	5
RegistryBrowserLabelProvider	4	129	4
Color	12	72	14
Cursor	4	24	5
Device	20	252	27
Font	9	62	11
FontData	14	89	14
FontMetrics	8	41	9
Region	15	76	16
Canvas	7	98	14
Caret	19	221	29
ColorDialog	5	39	5
DirectoryDialog	7	64	7
FileDialog	12	123	13
Group	6	64	11
Label	12	249	25
MessageBox	5	59	6
ProgressBar	8	78	14
Sash	4	150	13
Scale	14	119	18
Scrollable	6	185	19
TabItem	11	89	13
ToolBar	13	237	27

TABLE A- 6 Eclipse Class Attributes V2.0.2

Eclipse Class Attributes			
Class Name	**Version 2.0.2**		
	P-NOM	**LOC**	**NOM**
JDTCompilerAdapter	2	117	3
ClasspathDirectory	4	93	8
FileFinder	2	36	2
CodeFormatter	15	1872	50
AbstractMethodDeclaration	22	297	24
AbstractVariableDeclaration	3	24	3
Argument	5	62	5
ArrayAllocationExpression	6	126	6
ArrayInitializer	6	154	6
ClassFileReader	32	543	38
ClassFileStruct	16	161	16
ClassFormatException	3	44	3
FieldInfo	13	233	16
InnerClassInfo	7	89	8
MethodInfo	13	161	16
ConditionalFlowInfo	19	82	20
ExceptionHandlingFlowContext	6	142	7
FlowInfo	23	48	23
InitializationFlowContext	3	64	3
LabelFlowContext	3	35	4
LoopingFlowContext	7	121	9
BooleanConstant	5	22	5
ByteConstant	11	40	11
CompilerOptions	7	522	7
Constant	42	1254	42
IntConstant	11	41	11
LongConstant	11	40	11
StringConstant	5	21	5
RecoveredBlock	17	203	18
RecoveredField	12	117	12
RecoveredImport	7	31	7
RecoveredInitializer	12	172	12
RecoveredMethod	16	316	16
BufferedContent	9	53	9

Eclipse Class Attributes			
Class Name	**Version 2.0.2**		
	P-NOM	**LOC**	**NOM**
CompareConfiguration	27	174	27
CompareUI	12	66	13
CompareViewerPane	5	57	6
CompareViewerSwitchingPane	20	200	23
HistoryItem	6	30	6
NavigationAction	4	27	4
ResourceNode	14	92	14
AddFromHistoryAction	3	100	5
BinaryCompareViewer	4	94	5
ChangePropertyAction	4	31	4
CompareAction	2	24	2
CompareMessages	1	16	2
DocLineComparator	6	111	8
DocumentManager	4	42	4
ExceptionHandler	7	62	8
ImageMergeViewer	7	101	8
ResizableDialog	4	112	5
SimpleTextViewer	3	44	5
DiffContainer	7	51	7
DiffElement	7	32	7
DiffNode	23	175	24
DocumentRangeNode	17	175	19
DebugEvent	7	114	7
DebugException	1	13	1
Launch	23	194	25
Context	9	45	9
HrefUtil	4	54	4
Link	4	25	4
Toc	11	110	11
TocFile	13	74	13
TocFileParser	7	92	7
TocManager	5	136	9
URLCoder	2	49	4
OverlayIcon	8	95	8

Eclipse Class Attributes			
Class Name	**Version 2.0.2**		
	P-NOM	**LOC**	**NOM**
PDERuntimePlugin	10	67	12
PDERuntimePluginImages	2	107	5
RegistryBrowserLabelProvider	4	129	4
Color	12	72	14
Cursor	4	24	5
Device	20	252	27
Font	9	62	11
FontData	14	89	14
FontMetrics	8	41	9
Region	15	76	16
Canvas	7	98	14
Caret	19	221	29
ColorDialog	5	39	5
DirectoryDialog	7	64	7
FileDialog	12	123	13
Group	6	64	11
Label	12	249	25
MessageBox	5	59	6
ProgressBar	8	78	14
Sash	4	150	13
Scale	14	119	18
Scrollable	6	185	19
TabItem	11	89	13
ToolBar	13	237	27

TABLE A- 7 Eclipse Class Attributes V3.5

Eclipse Class Attributes			
Class Name	**Version 3.5**		
	P-NOM	**LOC**	**NOM**
JDTCompilerAdapter	3	392	6
ClasspathDirectory	10	151	13
FileFinder	1	28	2
CodeFormatter	3	20	3
AbstractMethodDeclaration	27	360	29
AbstractVariableDeclaration	11	78	11
Argument	9	132	9
ArrayAllocationExpression	5	126	5
ArrayInitializer	6	178	6
ClassFileReader	39	796	47
ClassFileStruct	10	71	10
ClassFormatException	9	94	9
FieldInfo	20	314	24
InnerClassInfo	6	82	7
MethodInfo	22	404	31
ConditionalFlowInfo	34	152	35
ExceptionHandlingFlowContext	10	204	10
FlowInfo	41	116	41
InitializationFlowContext	4	74	4
LabelFlowContext	3	30	4
LoopingFlowContext	12	411	13
BooleanConstant	5	24	6
ByteConstant	11	40	12
CompilerOptions	19	1310	19
Constant	34	1365	34
IntConstant	12	74	12
LongConstant	11	47	12
StringConstant	4	19	5
RecoveredBlock	21	324	23
RecoveredField	13	214	13
RecoveredImport	7	31	7
RecoveredInitializer	15	249	15
RecoveredMethod	21	514	22
BufferedContent	9	53	9

Eclipse Class Attributes			
Class Name	**Version 3.5**		
	P-NOM	**LOC**	**NOM**
CompareConfiguration	33	341	35
CompareUI	22	113	25
CompareViewerPane	20	170	21
CompareViewerSwitchingPane	13	192	16
HistoryItem	8	47	8
NavigationAction	4	47	4
ResourceNode	17	119	17
AddFromHistoryAction	3	104	5
BinaryCompareViewer	4	105	5
ChangePropertyAction	9	57	9
CompareAction	3	34	3
CompareMessages	0	122	1
DocLineComparator	6	110	8
DocumentManager	4	42	4
ExceptionHandler	7	62	8
ImageMergeViewer	7	100	8
ResizableDialog	6	122	7
SimpleTextViewer	3	42	5
DiffContainer	7	49	7
DiffElement	7	31	7
DiffNode	24	178	25
DocumentRangeNode	25	213	28
DebugEvent	9	129	9
DebugException	1	15	1
Launch	36	333	38
Context	12	121	12
HrefUtil	5	70	5
Link	4	21	4
Toc	16	136	18
TocFile	8	47	8
TocFileParser	1	32	1
TocManager	10	253	16
URLCoder	2	50	4
OverlayIcon	8	95	8

Eclipse Class Attributes			
Class Name	**Version 3.5**		
	P-NOM	**LOC**	**NOM**
PDERuntimePlugin	14	108	15
PDERuntimePluginImages	2	97	5
RegistryBrowserLabelProvider	6	325	6
Color	11	78	14
Cursor	8	395	11
Device	22	405	30
Font	9	208	15
FontData	14	164	17
FontMetrics	8	41	9
Region	24	177	28
Canvas	8	210	20
Caret	18	258	28
ColorDialog	5	55	5
DirectoryDialog	7	106	7
FileDialog	16	295	19
Group	6	85	11
Label	8	197	17
MessageBox	5	157	7
ProgressBar	10	79	13
Sash	4	295	18
Scale	14	138	18
Scrollable	6	273	29
TabItem	10	152	17
ToolBar	10	297	30

TABLE A- 8 Eclipse Class Attributes V3.6

Eclipse Class Attributes			
Class Name	**Version 3.6**		
	P-NOM	**LOC**	**NOM**
JDTCompilerAdapter	3	392	6
ClasspathDirectory	10	157	13
FileFinder	1	28	2
CodeFormatter	3	20	3
AbstractMethodDeclaration	27	364	29
AbstractVariableDeclaration	11	78	11
Argument	9	140	9
ArrayAllocationExpression	5	126	5
ArrayInitializer	6	178	6
ClassFileReader	39	801	47
ClassFileStruct	10	71	10
ClassFormatException	9	94	9
FieldInfo	20	314	24
InnerClassInfo	6	82	7
MethodInfo	22	404	31
ConditionalFlowInfo	36	160	37
ExceptionHandlingFlowContext	10	204	10
FlowInfo	44	172	44
InitializationFlowContext	4	74	4
LabelFlowContext	3	30	4
LoopingFlowContext	12	550	13
BooleanConstant	5	24	6
ByteConstant	11	40	12
CompilerOptions	18	1332	18
Constant	34	1365	34
IntConstant	12	74	12
LongConstant	11	47	12
StringConstant	4	19	5
RecoveredBlock	21	324	23
RecoveredField	14	234	14
RecoveredImport	7	31	7
RecoveredInitializer	15	249	15
RecoveredMethod	21	514	22
BufferedContent	9	53	9

Eclipse Class Attributes			
Class Name	**Version 3.6**		
	P-NOM	**LOC**	**NOM**
CompareConfiguration	33	341	35
CompareUI	22	113	25
CompareViewerPane	20	170	21
CompareViewerSwitchingPane	13	200	16
HistoryItem	8	47	8
NavigationAction	4	47	4
ResourceNode	17	119	17
AddFromHistoryAction	3	104	5
BinaryCompareViewer	4	105	5
ChangePropertyAction	9	57	9
CompareAction	3	34	3
CompareMessages	0	115	1
DocLineComparator	6	110	8
DocumentManager	4	42	4
ExceptionHandler	7	62	8
ImageMergeViewer	7	100	8
ResizableDialog	6	122	7
SimpleTextViewer	3	42	5
DiffContainer	7	49	7
DiffElement	7	31	7
DiffNode	24	179	25
DocumentRangeNode	25	213	28
DebugEvent	9	129	9
DebugException	1	15	1
Launch	36	333	38
Context	12	121	12
HrefUtil	5	70	5
Link	4	21	4
Toc	18	154	20
TocFile	8	47	8
TocFileParser	1	36	1
TocManager	10	261	16
URLCoder	3	66	6
OverlayIcon	8	95	8

Eclipse Class Attributes			
Class Name	**Version 3.6**		
	P-NOM	**LOC**	**NOM**
PDERuntimePlugin	14	108	15
PDERuntimePluginImages	2	97	5
RegistryBrowserLabelProvider	6	325	6
Color	11	78	14
Cursor	8	395	11
Device	22	405	30
Font	9	208	15
FontData	14	164	17
FontMetrics	8	41	9
Region	24	177	28
Canvas	8	207	21
Caret	18	258	28
ColorDialog	5	58	5
DirectoryDialog	7	109	7
FileDialog	16	329	19
Group	6	85	11
Label	8	225	17
MessageBox	5	160	7
ProgressBar	10	79	13
Sash	4	295	18
Scale	14	138	18
Scrollable	6	278	30
TabItem	10	152	17
ToolBar	10	324	32

TABLE A- 9 NetBeans Class Attributes V3.5.1

NetBeans Class Attributes			
Class Name	**Version 3.5.1**		
	P-NOM	**LOC**	**NOM**
IDESettingsBeanInfo	2	84	2
NbPlaces	19	117	21
NonGuiMain	11	364	16
Plain	8	65	8
WarmUpSupport	1	40	2
AboutAction	4	20	4
ConfigureShortcutsAction	4	31	4
GlobalPropertiesAction	5	28	5
HTMLViewAction	4	73	4
SystemExit	5	24	5
ViewRuntimeTabAction	5	26	5
DefaultParser	16	107	16
XMLEnvironmentProvider	3	36	3
PerformanceEvent	7	33	7
FileStateEditor	5	60	5
XML	2	19	2
FormatterIndentEngine	13	107	16
NbEditorUtilities	9	129	10
AnnotationTypesFolder	3	131	4
FontsColorsMIMEOptionFile	3	182	3
MIMEOptionFolder	5	265	10
OptionUtilities	23	368	25
RADVisualContainer	17	187	20
EventCustomEditor	2	241	8
FormDataLoader	7	85	7
MethodPicker	2	195	14
PersistenceManager	6	68	10
PropertyPicker	2	189	13
RADContainer	7	38	7
AddAction	6	88	6
CustomizeLayoutAction	7	52	7
EditContainerAction	4	45	4
EditFormAction	4	45	4
InPlaceEditAction	4	38	4

NetBeans Class Attributes			
Class Name	**Version 3.5.1**		
	P-NOM	**LOC**	**NOM**
InstallBeanAction	4	22	4
InstallToPaletteAction	7	53	7
ReloadAction	6	34	6
TestAction	7	94	7
CustomCodeEditor	2	95	7
EnumEditor	6	110	7
StringArrayEditor	18	127	18
UnknownLayoutSupport	3	11	3
GridLayoutSupport	4	106	4
JToolBarSupport	3	72	3
NullLayoutSupport	8	131	9
ScrollPaneSupport	5	63	6
PaletteItem	15	167	17
ScrollPopupMenu	6	100	7
SearchPanel	7	199	15
SearchTask	3	46	3
TextDetail	10	112	11
AntProjectDataLoader	7	114	8
ShortcutIterator	14	211	19
EventSetPatternNode	13	153	14
IdxPropertyPatternNode	9	118	10
Pattern	8	37	10
PropertyActionSettings	19	114	19
PropertyPatternNode	13	184	16
ClassElementFinder	3	181	8
DocumentModelBuilder	19	127	19
JavaModule	3	45	3
NodeFactoryPool	1	97	8
ParserAnnotation	7	110	14
ParserMessage	4	12	4
ContextDisplay	2	174	4
CookieDisplay	2	127	2
DispatchData	34	411	35
DisplayTable	19	191	24

NetBeans Class Attributes			
Class Name	**Version 3.5.1**		
	P-NOM	**LOC**	**NOM**
EditPanel	6	291	11
RequestData	42	431	43
TransactionNode	17	176	19
BundleNodeCustomizer	1	208	11
KeyNode	14	217	17
LocaleNodeCustomizer	1	277	13
PresentableFileEntry	24	165	28
PropertiesStructure	10	167	16
PropertyBundleEvent	7	54	7
PropertyBundleSupport	8	40	8
UtilConvert	14	352	18
AutoupdateType	12	71	12
CertificateDialog	1	129	5
SetupPanel	3	124	6
Access	6	62	7
ClassName	12	134	15
Code	6	74	10
ConstantPoolReader	16	159	18
CPInterfaceMethodInfo	1	9	2
Field	13	97	16
Method	4	58	7

TABLE A- 10 NetBeans Class Attributes V4.0

NetBeans Class Attributes			
Class Name	**Version 4.0**		
	P-NOM	**LOC**	**NOM**
IDESettingsBeanInfo	2	70	2
NbPlaces	19	138	21
NonGuiMain	11	365	16
Plain	8	62	8
WarmUpSupport	1	60	2
AboutAction	5	27	5
ConfigureShortcutsAction	6	37	6
GlobalPropertiesAction	5	26	5
HTMLViewAction	6	78	6
SystemExit	5	24	5
ViewRuntimeTabAction	3	27	3
DefaultParser	16	111	16
XMLEnvironmentProvider	3	36	3
PerformanceEvent	7	33	7
FileStateEditor	5	70	5
XML	2	19	2
FormatterIndentEngine	14	112	17
NbEditorUtilities	11	153	12
AnnotationTypesFolder	3	142	4
FontsColorsMIMEOptionFile	3	182	3
MIMEOptionFolder	5	266	10
OptionUtilities	27	383	29
RADVisualContainer	17	186	20
EventCustomEditor	2	239	8
FormDataLoader	6	54	6
MethodPicker	2	192	14
PersistenceManager	6	69	10
PropertyPicker	2	186	13
RADContainer	7	38	7
AddAction	7	90	8
CustomizeLayoutAction	8	57	8
EditContainerAction	5	51	5
EditFormAction	5	51	5
InPlaceEditAction	5	49	5

NetBeans Class Attributes			
Class Name	**Version 4.0**		
	P-NOM	**LOC**	**NOM**
InstallBeanAction	4	22	4
InstallToPaletteAction	6	30	6
ReloadAction	5	31	5
TestAction	8	97	8
CustomCodeEditor	2	95	7
EnumEditor	6	118	7
StringArrayEditor	18	127	18
UnknownLayoutSupport	3	11	3
GridLayoutSupport	4	105	4
JToolBarSupport	3	72	3
NullLayoutSupport	8	130	9
ScrollPaneSupport	5	73	6
PaletteItem	12	167	23
ScrollPopupMenu	7	116	8
SearchPanel	11	220	18
SearchTask	2	86	9
TextDetail	11	118	12
AntProjectDataLoader	5	57	6
ShortcutIterator	9	141	15
EventSetPatternNode	12	150	13
IdxPropertyPatternNode	8	115	9
Pattern	8	37	10
PropertyActionSettings	7	42	7
PropertyPatternNode	12	181	15
ClassElementFinder	3	214	10
DocumentModelBuilder	18	58	18
JavaModule	3	38	3
NodeFactoryPool	1	129	9
ParserAnnotation	7	126	15
ParserMessage	4	12	4
ContextDisplay	2	174	4
CookieDisplay	2	150	2
DispatchData	34	411	35
DisplayTable	21	212	26

NetBeans Class Attributes			
Class Name	**Version 4.0**		
	P-NOM	**LOC**	**NOM**
EditPanel	2	264	10
RequestData	42	434	43
TransactionNode	16	167	18
BundleNodeCustomizer	1	207	11
KeyNode	14	212	17
LocaleNodeCustomizer	1	277	13
PresentableFileEntry	24	151	27
PropertiesStructure	10	200	16
PropertyBundleEvent	7	64	7
PropertyBundleSupport	8	44	8
UtilConvert	11	133	14
AutoupdateType	12	76	12
CertificateDialog	1	128	5
SetupPanel	3	126	6
Access	6	66	7
ClassName	12	146	18
Code	8	95	12
ConstantPoolReader	16	159	18
CPInterfaceMethodInfo	1	9	2
Field	18	143	22
Method	13	134	15

TABLE A- 11 NetBeans Class Attributes V5.0

NetBeans Class Attributes			
Class Name	**Version 5.0**		
	P-NOM	**LOC**	**NOM**
IDESettingsBeanInfo	2	80	2
NbPlaces	11	97	13
NonGuiMain	11	366	16
Plain	8	62	8
WarmUpSupport	1	60	2
AboutAction	5	30	5
ConfigureShortcutsAction	6	23	6
GlobalPropertiesAction	5	26	5
HTMLViewAction	6	78	6
SystemExit	6	28	6
ViewRuntimeTabAction	3	26	3
DefaultParser	16	111	16
XMLEnvironmentProvider	3	36	3
PerformanceEvent	7	33	7
FileStateEditor	5	70	5
XML	2	19	2
FormatterIndentEngine	14	115	17
NbEditorUtilities	13	182	14
AnnotationTypesFolder	3	150	4
FontsColorsMIMEOptionFile	3	190	3
MIMEOptionFolder	5	276	10
OptionUtilities	27	382	29
RADVisualContainer	17	239	23
EventCustomEditor	2	239	8
FormDataLoader	6	46	6
MethodPicker	2	236	16
PersistenceManager	6	69	10
PropertyPicker	2	227	15
RADContainer	7	38	7
AddAction	7	90	8
CustomizeLayoutAction	7	54	7
EditContainerAction	5	50	5
EditFormAction	5	50	5
InPlaceEditAction	5	47	5

NetBeans Class Attributes			
Class Name	**Version 5.0**		
	P-NOM	**LOC**	**NOM**
InstallBeanAction	5	25	5
InstallToPaletteAction	7	33	7
ReloadAction	5	31	5
TestAction	8	117	8
CustomCodeEditor	2	96	7
EnumEditor	6	118	7
StringArrayEditor	18	156	18
UnknownLayoutSupport	2	9	2
GridLayoutSupport	4	118	4
JToolBarSupport	4	87	4
NullLayoutSupport	8	130	9
ScrollPaneSupport	5	73	6
PaletteItem	12	171	23
ScrollPopupMenu	7	122	8
SearchPanel	11	261	20
SearchTask	3	98	10
TextDetail	11	123	12
AntProjectDataLoader	6	37	6
ShortcutIterator	9	141	15
EventSetPatternNode	10	129	11
IdxPropertyPatternNode	8	102	9
Pattern	8	49	10
PropertyActionSettings	6	39	6
PropertyPatternNode	11	171	14
ClassElementFinder	3	214	10
DocumentModelBuilder	18	58	18
JavaModule	2	13	2
NodeFactoryPool	1	129	9
ParserAnnotation	7	126	15
ParserMessage	4	12	4
ContextDisplay	2	174	4
CookieDisplay	2	150	2
DispatchData	34	411	35
DisplayTable	21	212	26

NetBeans Class Attributes			
Class Name	**Version 5.0**		
	P-NOM	**LOC**	**NOM**
EditPanel	2	257	10
RequestData	42	434	43
TransactionNode	18	232	20
BundleNodeCustomizer	1	207	11
KeyNode	14	212	17
LocaleNodeCustomizer	1	277	13
PresentableFileEntry	24	151	27
PropertiesStructure	10	200	16
PropertyBundleEvent	7	64	7
PropertyBundleSupport	8	44	8
UtilConvert	11	133	14
AutoupdateType	12	78	12
CertificateDialog	1	128	5
SetupPanel	3	126	6
Access	6	66	7
ClassName	12	146	18
Code	9	109	13
ConstantPoolReader	16	159	18
CPInterfaceMethodInfo	1	9	2
Field	18	143	22
Method	13	134	15

TABLE A- 12 NetBeans Class Attributes V5.5.1

NetBeans Class Attributes			
Class Name	**Version 5.5.1**		
	P-NOM	**LOC**	**NOM**
IDESettingsBeanInfo	2	86	2
NbPlaces	11	97	13
NonGuiMain	11	366	16
Plain	8	62	8
WarmUpSupport	1	60	2
AboutAction	5	113	7
ConfigureShortcutsAction	6	23	6
GlobalPropertiesAction	5	26	5
HTMLViewAction	6	78	6
SystemExit	6	28	6
ViewRuntimeTabAction	3	26	3
DefaultParser	16	111	16
XMLEnvironmentProvider	3	36	3
PerformanceEvent	7	33	7
FileStateEditor	5	70	5
XML	2	19	2
FormatterIndentEngine	14	115	17
NbEditorUtilities	13	182	14
AnnotationTypesFolder	3	150	4
FontsColorsMIMEOptionFile	3	190	3
MIMEOptionFolder	5	276	10
OptionUtilities	27	382	29
RADVisualContainer	17	279	23
EventCustomEditor	2	239	8
FormDataLoader	6	46	6
MethodPicker	2	236	16
PersistenceManager	6	69	10
PropertyPicker	2	227	15
RADContainer	7	38	7
AddAction	7	90	8
CustomizeLayoutAction	7	54	7
EditContainerAction	5	50	5
EditFormAction	5	50	5
InPlaceEditAction	5	47	5

NetBeans Class Attributes			
Class Name	**Version 5.5.1**		
	P-NOM	**LOC**	**NOM**
InstallBeanAction	5	25	5
InstallToPaletteAction	7	33	7
ReloadAction	5	31	5
TestAction	8	117	8
CustomCodeEditor	2	96	7
EnumEditor	6	118	7
StringArrayEditor	18	156	18
UnknownLayoutSupport	2	9	2
GridLayoutSupport	4	118	4
JToolBarSupport	4	87	4
NullLayoutSupport	8	130	9
ScrollPaneSupport	5	73	6
PaletteItem	12	171	23
ScrollPopupMenu	7	122	8
SearchPanel	11	261	20
SearchTask	3	98	10
TextDetail	11	123	12
AntProjectDataLoader	6	37	6
ShortcutIterator	9	141	15
EventSetPatternNode	10	129	11
IdxPropertyPatternNode	8	102	9
Pattern	8	49	10
PropertyActionSettings	6	39	6
PropertyPatternNode	11	171	14
ClassElementFinder	3	214	10
DocumentModelBuilder	18	58	18
JavaModule	2	13	2
NodeFactoryPool	1	129	9
ParserAnnotation	7	126	15
ParserMessage	4	12	4
ContextDisplay	2	174	4
CookieDisplay	2	150	2
DispatchData	34	411	35
DisplayTable	21	212	26

NetBeans Class Attributes			
Class Name	**Version 5.5.1**		
	P-NOM	**LOC**	**NOM**
EditPanel	2	257	10
RequestData	42	434	43
TransactionNode	18	232	20
BundleNodeCustomizer	1	207	11
KeyNode	14	212	17
LocaleNodeCustomizer	1	277	13
PresentableFileEntry	24	151	27
PropertiesStructure	10	200	16
PropertyBundleEvent	7	64	7
PropertyBundleSupport	8	44	8
UtilConvert	11	133	14
AutoupdateType	12	81	12
CertificateDialog	1	128	5
SetupPanel	3	126	6
Access	6	66	7
ClassName	12	146	18
Code	9	109	13
ConstantPoolReader	16	159	18
CPInterfaceMethodInfo	1	9	2
Field	18	143	22
Method	13	134	15

Appendix B

Class Stability Metrics Measurement Results

TABLE B- 1 Android Class Stability Measurement (CII)

Android Class Stability Measurement			
Class Name	**CII (LOC)**		
	1.5/1.6	**1.6/2.1**	**2.1/2.3.1**
Typeface	10.60606	1.369863	4.054054
Scroller	0	2.051282	-0.50251
DeleteEventHelper	0	-0.52632	0.529101
OnScreenHint	-19.7452	-11.1111	-0.89286
EmailAddressValidator	0	0	-15.3846
MessagingListener	14.28571	-39.5833	-3.44828
AlarmManager	0	0	11.32
NotificationManager	0	12.9	1.42
StatusBarManager	0	0	-7.25
AppWidgetProvider	-4.54545	0	0
ActivityNotFoundException	0	0	0
ContentUris	0	0	0
MutableContextWrapper	0	0	0
ReceiverCallNotAllowedException	0	0	0
SyncContext	0	3.030303	11.76471
TypedArray	0.268817	0	5.36193
CharArrayBuffer	0	0	0
CursorIndexOutOfBoundsException	0	0	0
DataSetObserver	0	0	0
SQLException	0	0	0
StaleDataException	0	0	0
SQLiteAbortException	0	0	0
SQLiteClosable	0	0	59.375
SQLiteConstraintException	0	0	0
SQLiteDatabaseCorruptException	0	0	0
SQLiteDiskIOException	0	0	0
SQLiteDoneException	0	0	0
SQLiteException	0	0	0
SQLiteFullException	0	0	0
SQLiteMisuseException	0	0	0
SQLiteOpenHelper	0	0	3.636364
SQLiteStatement	0	0	-15.9574
GpsSatellite	0	0	0
LocationProvider	68.96552	0	-18.3673

Android Class Stability Measurement			
Class Name	**CII (LOC)**		
	1.5/1.6	**1.6/2.1**	**2.1/2.3.1**
Ringtone	0	0	6.711409
FocusFinderHelper	0	0	0
Gravity	0	0	0
InflateException	0	0	0
SoundEffectConstants	0	0	0
TouchDelegate	0	0	0
WindowManagerImpl	0	0	0
AccelerateInterpolator	20	0	0
BaseInputConnection	3.076923	0	13.93035
DocumentBuilder	0	0	49.12281
FactoryConfigurationError	0	0	-3.22581
ParserConfigurationException	0	0	0
SAXParser	0	0	23.25581
AbstractMethodError	0	0	0
ArrayStoreException	0	0	0
AssertionError	0	0	0
ClassFormatError	0	0	0
Error	0	0	0
Exception	0	0	0
IllegalAccessError	0	0	0
IllegalAccessException	0	0	0
IllegalStateException	0	0	0
Math	0	8.917197	136.8421
Number	0	0	0
OutOfMemoryError	0	0	0
Process	0	0	0
RuntimeException	0	0	0
RuntimePermission	0	0	0
SecurityException	0	0	0
StackOverflowError	0	0	0
StrictMath	0	0	61.71429
StringIndexOutOfBoundsException	0	0	-7.14286
ThreadDeath	0	0	0
TypeNotPresentException	0	0	0

Android Class Stability Measurement			
Class Name	**CII (LOC)**		
	1.5/1.6	**1.6/2.1**	**2.1/2.3.1**
UnknownError	0	0	0
VerifyError	0	0	0
AlgorithmParameterGenerator	0	0	-2.43902
AuthProvider	0	0	0
CollationElementIterator	0	0	0
Environment	0	0	175
SystemProperties	0	-21.2121	0
Vibrator	0	2.857143	44.44444

TABLE B- 2 Android Class Stability Measurement (Grosser)

Android Class Stability Measurement			
Class Name	**Grosser(P-NOM)**		
	1.5/1.6	**1.6/2.1**	**2.1/2.3.1**
Typeface	1	1	1
Scroller	1	1	1
DeleteEventHelper	1	1	1
OnScreenHint	0.56	0.56	1
EmailAddressValidator	1	1	1
MessagingListener	1	0.56	1
AlarmManager	1	1	1
NotificationManager	1	1	1
StatusBarManager	1	1	0.57
AppWidgetProvider	1	1	1
ActivityNotFoundException	1	1	1
ContentUris	1	1	1
MutableContextWrapper	1	1	1
ReceiverCallNotAllowedException	1	1	1
SyncContext	1	0.6	1
TypedArray	1	1	1
CharArrayBuffer	1	1	1

Android Class Stability Measurement			
Class Name	Grosser(P-NOM)		
	1.5/1.6	1.6/2.1	2.1/2.3.1
CursorIndexOutOfBoundsException	1	1	1
DataSetObserver	1	1	1
SQLException	1	1	1
StaleDataException	1	1	1
SQLiteAbortException	1	1	1
SQLiteClosable	1	1	1
SQLiteConstraintException	1	1	1
SQLiteDatabaseCorruptException	1	1	1
SQLiteDiskIOException	1	1	1
SQLiteDoneException	1	1	1
SQLiteException	1	1	1
SQLiteFullException	1	1	1
SQLiteMisuseException	1	1	1
SQLiteOpenHelper	1	1	1
SQLiteStatement	1	1	1
GpsSatellite	1	1	1
LocationProvider	1	1	1
Ringtone	1	1	1
FocusFinderHelper	1	1	1
Gravity	1	1	1
InflateException	1	1	1
SoundEffectConstants	1	1	1
TouchDelegate	1	1	1
WindowManagerImpl	1	1	1
AccelerateInterpolator	1	1	1
BaseInputConnection	1	1	1
DocumentBuilder	1	1	0.79
FactoryConfigurationError	1	1	1
ParserConfigurationException	1	1	1
SAXParser	0.89	1	1
AbstractMethodError	1	1	1
ArrayStoreException	1	1	1
AssertionError	1	1	1
ClassFormatError	1	1	1

Android Class Stability Measurement			
Class Name	**Grosser(P-NOM)**		
	1.5/1.6	**1.6/2.1**	**2.1/2.3.1**
Error	1	1	1
Exception	1	1	1
IllegalAccessError	1	1	1
IllegalAccessException	1	1	1
IllegalStateException	1	1	1
Math	1	1	1
Number	1	1	1
OutOfMemoryError	1	1	1
Process	1	1	1
RuntimeException	1	1	1
RuntimePermission	1	1	1
SecurityException	1	1	1
StackOverflowError	1	1	1
StrictMath	1	1	1
StringIndexOutOfBoundsException	1	1	1
ThreadDeath	1	1	1
TypeNotPresentException	1	1	1
UnknownError	1	1	1
VerifyError	1	1	1
AlgorithmParameterGenerator	1	1	1
AuthProvider	1	1	1
CollationElementIterator	1	1	1
Environment	1	1	1
SystemProperties	1	1	1
Vibrator	1	1	1

TABLE B- 3 Android Class Stability Measurement (Stab)

Android Class Stability Measurement			
Class Name	**Stab (NOM)**		
	1.5/1.6	**1.6/2.1**	**2.1/2.3.1**
Typeface	0	0	1
Scroller	1	0	1
DeleteEventHelper	1	1	1
OnScreenHint	0	0	1
EmailAddressValidator	1	1	1
MessagingListener	0	0	1
AlarmManager	1	1	0
NotificationManager	1	0	1
StatusBarManager	1	1	0
AppWidgetProvider	1	1	1
ActivityNotFoundException	1	1	1
ContentUris	1	1	1
MutableContextWrapper	1	1	1
ReceiverCallNotAllowedException	1	1	1
SyncContext	1	1	1
TypedArray	1	1	0
CharArrayBuffer	1	1	1
CursorIndexOutOfBoundsException	1	1	1
DataSetObserver	1	1	1
SQLException	1	1	1
StaleDataException	1	1	1
SQLiteAbortException	1	1	1
SQLiteClosable	1	1	0
SQLiteConstraintException	1	1	1
SQLiteDatabaseCorruptException	1	1	1
SQLiteDiskIOException	1	1	1
SQLiteDoneException	1	1	1
SQLiteException	1	1	1
SQLiteFullException	1	1	1
SQLiteMisuseException	1	1	1
SQLiteOpenHelper	1	1	1
SQLiteStatement	1	1	1
GpsSatellite	1	1	1
LocationProvider	1	1	1

Android Class Stability Measurement			
Class Name	**Stab (NOM)**		
	1.5/1.6	**1.6/2.1**	**2.1/2.3.1**
Ringtone	1	1	1
FocusFinderHelper	1	1	1
Gravity	1	1	1
InflateException	1	1	1
SoundEffectConstants	1	1	1
TouchDelegate	1	1	1
WindowManagerImpl	1	1	1
AccelerateInterpolator	1	1	1
BaseInputConnection	1	1	0
DocumentBuilder	1	1	0
FactoryConfigurationError	1	1	1
ParserConfigurationException	1	1	1
SAXParser	1	1	0
AbstractMethodError	1	1	1
ArrayStoreException	1	1	1
AssertionError	1	1	1
ClassFormatError	1	1	1
Error	1	1	1
Exception	1	1	1
IllegalAccessError	1	1	1
IllegalAccessException	1	1	1
IllegalStateException	1	1	1
Math	1	1	0
Number	1	1	1
OutOfMemoryError	1	1	1
Process	1	1	1
RuntimeException	1	1	1
RuntimePermission	1	1	1
SecurityException	1	1	1
StackOverflowError	1	1	1
StrictMath	1	1	0
StringIndexOutOfBoundsException	1	1	1
ThreadDeath	1	1	1
TypeNotPresentException	1	1	1

Android Class Stability Measurement			
Class Name	**Stab (NOM)**		
	1.5/1.6	**1.6/2.1**	**2.1/2.3.1**
UnknownError	1	1	1
VerifyError	1	1	1
AlgorithmParameterGenerator	1	1	1
AuthProvider	1	1	0
CollationElementIterator	1	1	0
Environment	1	1	0
SystemProperties	1	0	1
Vibrator	1	1	1

TABLE B- 4 Android Class Stability Measurement (CSM)

Android Class Stability Measurement			
Class Name	**CSM**		
	1.5/1.6	**1.5/2.1**	**1.5/2.3.1**
Typeface	0.973214	0.8125	0.732143
Scroller	1	0.8125	0.723545
DeleteEventHelper	1	0.804924	0.732008
OnScreenHint	0.777083	0.642708	0.60625
EmailAddressValidator	1	0.8125	0.75
MessagingListener	1	0.629464	0.628348
AlarmManager	1	0.8125	0.75
NotificationManager	1	0.775	0.725
StatusBarManager	1	0.8125	0.671875
AppWidgetProvider	0.979167	0.802083	0.743056
ActivityNotFoundException	1	0.8125	0.75
ContentUris	1	0.8125	0.75
MutableContextWrapper	1	0.8125	0.75
ReceiverCallNotAllowedException	1	0.8125	0.75
SyncContext	1	0.7625	0.7
TypedArray	0.995968	0.804436	0.744624
CharArrayBuffer	1	0.8125	0.75

Android Class Stability Measurement			
Class Name	**CSM**		
	1.5/1.6	**1.5/2.1**	**1.5/2.3.1**
CursorIndexOutOfBoundsException	1	0.8125	0.75
DataSetObserver	1	0.8125	0.75
SQLException	1	0.8125	0.75
StaleDataException	1	0.8125	0.75
SQLiteAbortException	1	0.8125	0.75
SQLiteClosable	1	0.8125	0.741667
SQLiteConstraintException	1	0.8125	0.75
SQLiteDatabaseCorruptException	1	0.8125	0.75
SQLiteDiskIOException	1	0.8125	0.75
SQLiteDoneException	1	0.8125	0.75
SQLiteException	1	0.8125	0.75
SQLiteFullException	1	0.8125	0.75
SQLiteMisuseException	1	0.8125	0.75
SQLiteOpenHelper	1	0.8125	0.75
SQLiteStatement	1	0.757813	0.463542
GpsSatellite	1	0.8125	0.75
LocationProvider	0.989583	0.807292	0.715278
Ringtone	1	0.8125	0.740385
FocusFinderHelper	0.875	0.75	0.708333
Gravity	1	0.8125	0.75
InflateException	1	0.8125	0.75
SoundEffectConstants	1	0.8125	0.75
TouchDelegate	1	0.8125	0.75
WindowManagerImpl	1	0.8125	0.75
AccelerateInterpolator	0.75	0.625	0.583333
BaseInputConnection	0.986111	0.805556	0.742284
DocumentBuilder	1	0.8125	0.720238
FactoryConfigurationError	1	0.8125	0.458333
ParserConfigurationException	1	0.8125	0.75
SAXParser	1	0.8125	0.719298
AbstractMethodError	1	0.8125	0.75
ArrayStoreException	1	0.8125	0.75
AssertionError	1	0.8125	0.75
ClassFormatError	1	0.8125	0.75

Android Class Stability Measurement			
Class Name	**CSM**		
	1.5/1.6	**1.5/2.1**	**1.5/2.3.1**
Error	1	0.8125	0.75
Exception	1	0.8125	0.75
IllegalAccessError	1	0.8125	0.75
IllegalAccessException	1	0.8125	0.75
IllegalStateException	1	0.8125	0.75
Math	0.989362	0.799202	0.737589
Number	1	0.8125	0.75
OutOfMemoryError	1	0.8125	0.75
Process	1	0.8125	0.75
RuntimeException	1	0.8125	0.75
RuntimePermission	1	0.8125	0.75
SecurityException	1	0.8125	0.71875
StackOverflowError	1	0.8125	0.75
StrictMath	1	0.808511	0.734043
StringIndexOutOfBoundsException	1	0.8125	0.736111
ThreadDeath	1	0.8125	0.75
TypeNotPresentException	1	0.8125	0.708333
UnknownError	1	0.8125	0.75
VerifyError	1	0.8125	0.75
AlgorithmParameterGenerator	1	0.8125	0.738636
AuthProvider	1	0.8125	0.71875
CollationElementIterator	1	0.8125	0.75
Environment	0.85	0.8125	0.684028
SystemProperties	1	0.784722	0.731482
Vibrator	1	0.765625	0.708333

TABLE B- 5 Eclipse Class Stability Measurement (CII)

Eclipse Class Stability Measurement			
Class Name	**CII (LOC)**		
	2.0/2.0.2	**2.0.2/3.5**	**3.5/3.6**
JDTCompilerAdapter	8.333333333	235.042735	0
ClasspathDirectory	0	62.3655914	3.973509934
FileFinder	0	-22.2222222	0
CodeFormatter	0	-98.9316239	0
AbstractMethodDeclaration	0	21.21212121	1.111111111
AbstractVariableDeclaration	0	225	0
Argument	0	112.9032258	6.060606061
ArrayAllocationExpression	0	0	0
ArrayInitializer	0	15.58441558	0
ClassFileReader	0	46.59300184	0.628140704
ClassFileStruct	0	-55.9006211	0
ClassFormatException	0	113.6363636	0
FieldInfo	0	34.7639485	0
InnerClassInfo	0	-7.86516854	0
MethodInfo	0	150.931677	0
ConditionalFlowInfo	0	85.36585366	5.263157895
ExceptionHandlingFlowContext	0	43.66197183	0
FlowInfo	0	141.6666667	48.27586207
InitializationFlowContext	0	15.625	0
LabelFlowContext	0	-14.2857143	0
LoopingFlowContext	0	239.6694215	33.81995134
BooleanConstant	0	9.090909091	0
ByteConstant	0	0	0
CompilerOptions	0	150.9578544	1.679389313
Constant	0	8.851674641	0
IntConstant	0	80.48780488	0
LongConstant	0	17.5	0
StringConstant	0	-9.52380952	0
RecoveredBlock	0	59.60591133	0
RecoveredField	0	82.90598291	9.345794393
RecoveredImport	0	0	0
RecoveredInitializer	0	44.76744186	0
RecoveredMethod	0	62.65822785	0
BufferedContent	0	0	0

Eclipse Class Stability Measurement			
Class Name	**CII (LOC)**		
	2.0/2.0.2	**2.0.2/3.5**	**3.5/3.6**
CompareConfiguration	0	95.97701149	0
CompareUI	0	71.21212121	0
CompareViewerPane	0	198.245614	0
CompareViewerSwitchingPane	0	-4	4.166666667
HistoryItem	0	56.66666667	0
NavigationAction	0	74.07407407	0
ResourceNode	0	29.34782609	0
AddFromHistoryAction	0	4	0
BinaryCompareViewer	0	11.70212766	0
ChangePropertyAction	0	83.87096774	0
CompareAction	0	41.66666667	0
CompareMessages	0	662.5	-5.73770492
DocLineComparator	0	-0.9009009	0
DocumentManager	0	0	0
ExceptionHandler	0	0	0
ImageMergeViewer	0	-0.99009901	0
ResizableDialog	0	8.928571429	0
SimpleTextViewer	0	-4.54545455	0
DiffContainer	0	-3.92156863	0
DiffElement	0	-3.125	0
DiffNode	0	1.714285714	0.561797753
DocumentRangeNode	0	21.71428571	0
DebugEvent	0	13.15789474	0
DebugException	0	15.38461538	0
Launch	0	71.64948454	0
Context	0	168.8888889	0
HrefUtil	0	29.62962963	0
Link	0	-16	0
Toc	0	23.63636364	13.23529412
TocFile	0	-36.4864865	0
TocFileParser	0	-65.2173913	12.5
TocManager	0	86.02941176	3.162055336
URLCoder	0	2.040816327	32
OverlayIcon	0	0	0

Eclipse Class Stability Measurement			
Class Name	**CII (LOC)**		
	2.0/2.0.2	**2.0.2/3.5**	**3.5/3.6**
PDERuntimePlugin	0	61.19402985	0
PDERuntimePluginImages	0	-9.34579439	0
RegistryBrowserLabelProvider	0	151.9379845	0
Color	0	8.333333333	0
Cursor	0	1545.833333	0
Device	0	60.71428571	0
Font	0	235.483871	0
FontData	0	84.26966292	0
FontMetrics	0	0	0
Region	0	132.8947368	0
Canvas	0	114.2857143	-1.42857143
Caret	0	16.74208145	0
ColorDialog	0	41.02564103	5.454545455
DirectoryDialog	0	65.625	2.830188679
FileDialog	0	139.8373984	11.52542373
Group	0	32.8125	0
Label	0	-20.8835341	14.21319797
MessageBox	0	166.1016949	1.910828025
ProgressBar	0	1.282051282	0
Sash	0	96.66666667	0
Scale	0	15.96638655	0
Scrollable	0	47.56756757	1.831501832
TabItem	0	70.78651685	0
ToolBar	0	25.3164557	9.090909091

TABLE B- 6 Eclipse Class Stability Measurement (Grosser)

Eclipse Class Stability Measurement			
Class Name	**Grosser(P-NOM)**		
	2.0/2.0.2	**2.0.2/3.5**	**3.5/3.6**
JDTCompilerAdapter	1	1	1
ClasspathDirectory	1	1	1
FileFinder	1	0.5	1
CodeFormatter	1	0	1
AbstractMethodDeclaration	1	1	1
AbstractVariableDeclaration	1	0	1
Argument	1	0.8	1
ArrayAllocationExpression	1	0.67	1
ArrayInitializer	1	0.83	1
ClassFileReader	1	0.91	1
ClassFileStruct	1	0.625	1
ClassFormatException	1	1	1
FieldInfo	1	0.66	1
InnerClassInfo	1	0.86	1
MethodInfo	1	0.92	1
ConditionalFlowInfo	1	0.79	1
ExceptionHandlingFlowContext	1	1	1
FlowInfo	1	0.74	1
InitializationFlowContext	1	1	1
LabelFlowContext	1	1	1
LoopingFlowContext	1	0.86	1
BooleanConstant	1	0.8	1
ByteConstant	1	0.91	1
CompilerOptions	1	0.43	0.95
Constant	1	0.79	1
IntConstant	1	1	1
LongConstant	1	0.91	1
StringConstant	1	0.6	1
RecoveredBlock	1	1	1
RecoveredField	1	1	1
RecoveredImport	1	1	1
RecoveredInitializer	1	1	1
RecoveredMethod	1	1	1
BufferedContent	1	1	1

Eclipse Class Stability Measurement			
Class Name	**Grosser(P-NOM)**		
	2.0/2.0.2	**2.0.2/3.5**	**3.5/3.6**
CompareConfiguration	1	1	1
CompareUI	1	1	1
CompareViewerPane	1	1	1
CompareViewerSwitchingPane	1	0.5	1
HistoryItem	1	1	1
NavigationAction	1	1	1
ResourceNode	1	1	1
AddFromHistoryAction	1	0.67	1
BinaryCompareViewer	1	1	1
ChangePropertyAction	1	1	1
CompareAction	1	0.5	1
CompareMessages	1	0	0
DocLineComparator	1	1	1
DocumentManager	1	1	1
ExceptionHandler	1	1	1
ImageMergeViewer	1	1	1
ResizableDialog	1	1	1
SimpleTextViewer	1	1	1
DiffContainer	1	1	1
DiffElement	1	1	1
DiffNode	1	1	1
DocumentRangeNode	1	1	1
DebugEvent	1	1	1
DebugException	1	1	1
Launch	1	1	1
Context	1	0.56	1
HrefUtil	1	1	1
Link	1	0.5	1
Toc	1	0.45	1
TocFile	1	0.46	1
TocFileParser	1	0	1
TocManager	1	0.4	1
URLCoder	1	1	1
OverlayIcon	1	1	1

Eclipse Class Stability Measurement			
Class Name	**Grosser(P-NOM)**		
	2.0/2.0.2	**2.0.2/3.5**	**3.5/3.6**
PDERuntimePlugin	1	0.6	1
PDERuntimePluginImages	1	1	1
RegistryBrowserLabelProvider	1	1	1
Color	1	0.92	1
Cursor	1	0.75	1
Device	1	1	1
Font	1	0.89	1
FontData	1	0.93	1
FontMetrics	1	1	1
Region	1	0.87	1
Canvas	1	0.57	1
Caret	1	0.95	1
ColorDialog	1	1	1
DirectoryDialog	1	1	1
FileDialog	1	1	1
Group	1	1	1
Label	1	0.67	1
MessageBox	1	1	1
ProgressBar	1	1	1
Sash	1	1	1
Scale	1	1	1
Scrollable	1	0.83	1
TabItem	1	0.91	1
ToolBar	1	0.77	1

TABLE B- 7 Eclipse Class Stability Measurement (Stab)

Eclipse Class Stability Measurement			
Class Name	**Stab (NOM)**		
	2.0/2.0.2	**2.0.2/3.5**	**3.5/3.6**
JDTCompilerAdapter	1	0	1
ClasspathDirectory	1	0	1
FileFinder	1	1	1
CodeFormatter	1	0	1
AbstractMethodDeclaration	1	0	1
AbstractVariableDeclaration	1	0	1
Argument	1	0	1
ArrayAllocationExpression	1	0	1
ArrayInitializer	1	1	1
ClassFileReader	1	0	1
ClassFileStruct	1	0	1
ClassFormatException	1	0	1
FieldInfo	1	0	1
InnerClassInfo	1	0	1
MethodInfo	1	0	1
ConditionalFlowInfo	1	0	0
ExceptionHandlingFlowContext	1	0	1
FlowInfo	1	0	0
InitializationFlowContext	1	0	1
LabelFlowContext	1	1	1
LoopingFlowContext	1	0	1
BooleanConstant	1	0	1
ByteConstant	1	0	1
CompilerOptions	1	0	0
Constant	1	0	1
IntConstant	1	0	1
LongConstant	1	0	1
StringConstant	1	1	1
RecoveredBlock	1	0	1
RecoveredField	1	0	0
RecoveredImport	1	1	1
RecoveredInitializer	1	0	1
RecoveredMethod	1	0	1
BufferedContent	1	1	1

Eclipse Class Stability Measurement			
Class Name	**Stab (NOM)**		
	2.0/2.0.2	**2.0.2/3.5**	**3.5/3.6**
CompareConfiguration	1	0	1
CompareUI	1	0	1
CompareViewerPane	1	0	1
CompareViewerSwitchingPane	1	0	1
HistoryItem	1	0	1
NavigationAction	1	1	1
ResourceNode	1	0	1
AddFromHistoryAction	1	1	1
BinaryCompareViewer	1	1	1
ChangePropertyAction	1	0	1
CompareAction	1	0	1
CompareMessages	1	0	1
DocLineComparator	1	1	1
DocumentManager	1	1	1
ExceptionHandler	1	1	1
ImageMergeViewer	1	1	1
ResizableDialog	1	0	1
SimpleTextViewer	1	1	1
DiffContainer	1	1	1
DiffElement	1	1	1
DiffNode	1	0	1
DocumentRangeNode	1	0	1
DebugEvent	1	0	1
DebugException	1	1	1
Launch	1	0	1
Context	1	0	1
HrefUtil	1	0	1
Link	1	1	1
Toc	1	0	0
TocFile	1	0	1
TocFileParser	1	0	1
TocManager	1	0	1
URLCoder	1	1	0
OverlayIcon	1	1	1

Eclipse Class Stability Measurement			
Class Name	**Stab (NOM)**		
	2.0/2.0.2	**2.0.2/3.5**	**3.5/3.6**
PDERuntimePlugin	1	0	1
PDERuntimePluginImages	1	1	1
RegistryBrowserLabelProvider	1	0	1
Color	1	1	1
Cursor	1	0	1
Device	1	0	1
Font	1	0	1
FontData	1	0	1
FontMetrics	1	1	1
Region	1	0	1
Canvas	1	0	0
Caret	1	0	1
ColorDialog	1	1	1
DirectoryDialog	1	1	1
FileDialog	1	0	1
Group	1	1	1
Label	1	0	1
MessageBox	1	0	1
ProgressBar	1	0	1
Sash	1	0	1
Scale	1	1	1
Scrollable	1	0	0
TabItem	1	0	1
ToolBar	1	0	0

TABLE B- 8 Eclipse Class Stability Measurement (CSM)

Eclipse Class Stability Measurement			
Class Name	**CSM**		
	2.0/2.0.2	**2.0/3.5**	**2.0/3.6**
JDTCompilerAdapter	0.9166667	0.7083334	0.6805556
ClasspathDirectory	1	0.43861606	0.39955357
FileFinder	1	0.375	0.375
CodeFormatter	1	0.125	0.125
AbstractMethodDeclaration	0.9947917	0.7005208	0.67013896
AbstractVariableDeclaration	1	0.625	0.625
Argument	1	0.68749994	0.6666666
ArrayAllocationExpression	1	0.625	0.6041666
ArrayInitializer	1	0.71875006	0.6875
ClassFileReader	1	0.62345195	0.5773994
ClassFileStruct	0.9765625	0.5703125	0.546875
ClassFormatException	1	0.75	0.7083334
FieldInfo	1	0.434375	0.39791667
InnerClassInfo	1	0.72727275	0.68560606
MethodInfo	1	0.52225375	0.48579547
ConditionalFlowInfo	1	0.71875	0.68749994
ExceptionHandlingFlowContext	1	0.6765873	0.6602182
FlowInfo	1	0.48369566	0.44746375
InitializationFlowContext	1	0.75000006	0.7083334
LabelFlowContext	1	0.71875	0.6875
LoopingFlowContext	1	0.7083333	0.6805555
BooleanConstant	1	0.6375	0.5916667
ByteConstant	1	0.6306818	0.58712125
CompilerOptions	1	0.5027985	0.48196515
Constant	1	0.48809522	0.47123015
IntConstant	1	0.7992424	0.6903409
LongConstant	1	0.6306818	0.58712125
StringConstant	1	0.58750004	0.55833334
RecoveredBlock	1	0.5972222	0.5648148
RecoveredField	1	0.7343475	0.6979167
RecoveredImport	1	0.74999994	0.7083333
RecoveredInitializer	1	0.59375006	0.5625
RecoveredMethod	1	0.61328125	0.5755209
BufferedContent	1	0.6666667	0.61111116

Eclipse Class Stability Measurement			
Class Name	**CSM**		
	2.0/2.0.2	**2.0/3.5**	**2.0/3.6**
CompareConfiguration	1	0.7320907	0.67884994
CompareUI	1	0.7932692	0.7371795
CompareViewerPane	1	0.7708333	0.72222227
CompareViewerSwitchingPane	1	0.44065657	0.41414142
HistoryItem	1	0.7291666	0.6666667
NavigationAction	1	0.78125	0.7291667
ResourceNode	1	0.7410714	0.68154764
AddFromHistoryAction	1	0.475	0.4416667
BinaryCompareViewer	1	0.7333333	0.7333333
ChangePropertyAction	1	0.796875	0.7395834
CompareAction	1	0.375	0.375
CompareMessages	1	0.34375	0.31250003
DocLineComparator	1	0.78125	0.7291667
DocumentManager	1	0.8125	0.75000006
ExceptionHandler	1	0.8125	0.75000006
ImageMergeViewer	1	0.76116073	0.7038691
ResizableDialog	1	0.8	0.74166673
SimpleTextViewer	1	0.775	0.725
DiffContainer	1	0.8125	0.75000006
DiffElement	1	0.8125	0.75000006
DiffNode	1	0.8098958	0.62326396
DocumentRangeNode	1	0.712641	0.6611059
DebugEvent	1	0.79464287	0.7380953
DebugException	1	0.8125	0.75000006
Launch	1	0.625	0.5833333
Context	1	0.2916667	0.2777778
HrefUtil	1	0.71875	0.6875
Link	1	0.265625	0.26041666
Toc	1	0.42045456	0.280303
TocFile	1	0.45398355	0.43956047
TocFileParser	1	0.25892857	0.2559524
TocManager	1	0.3888889	0.38425925
URLCoder	1	0.796875	0.6979167
OverlayIcon	1	0.7578125	0.7135417

Eclipse Class Stability Measurement			
Class Name	**CSM**		
	2.0/2.0.2	**2.0/3.5**	**2.0/3.6**
PDERuntimePlugin	1	0.5885417	0.55902773
PDERuntimePluginImages	1	0.64855075	0.6068841
RegistryBrowserLabelProvider	1	0.46875	0.4375
Color	1	0.59375	0.5625
Cursor	1	0.58750004	0.5583333
Device	0.9972826	0.61654586	0.5795089
Font	1	0.5852273	0.5568181
FontData	1	0.5625	0.51488096
FontMetrics	0.87499994	0.7152778	0.6972222
Region	1	0.7265625	0.6927083
Canvas	1	0.68749994	0.6666667
Caret	1	0.71532136	0.67006266
ColorDialog	1	0.65	0.6
DirectoryDialog	1	0.77678573	0.7261905
FileDialog	1	0.7240384	0.674359
Group	1	0.46590906	0.43560606
Label	1	0.55999994	0.53999996
MessageBox	1	0.7708333	0.72222227
ProgressBar	1	0.70982146	0.68154764
Sash	1	0.65064096	0.6282051
Scale	1	0.49305558	0.45370367
Scrollable	1	0.62039465	0.5885965
TabItem	1	0.75000006	0.7083333
ToolBar	1	0.59259254	0.5617284

TABLE B- 9 NetBeans Class Stability Measurement (CII)

NetBeans Class Stability Measurement			
Class Name	**CII (LOC)**		
	3.5.1/4.0	**4.0/5.0**	**5.0/5.5.1**
IDESettingsBeanInfo	-16.6666667	14.2857143	7.5
NbPlaces	17.9487179	-29.7101449	0
NonGuiMain	0.27472527	0.2739726	0
Plain	-4.61538462	0	0
WarmUpSupport	50	0	0
AboutAction	35	11.1111111	276.666667
ConfigureShortcutsAction	19.3548387	-37.8378378	0
GlobalPropertiesAction	-7.14285714	0	0
HTMLViewAction	6.84931507	0	0
SystemExit	0	16.6666667	0
ViewRuntimeTabAction	3.84615385	-3.7037037	0
DefaultParser	3.73831776	0	0
XMLEnvironmentProvider	0	0	0
PerformanceEvent	0	0	0
FileStateEditor	16.6666667	0	0
XML	0	0	0
FormatterIndentEngine	4.6728972	2.67857143	0
NbEditorUtilities	18.6046512	18.9542484	0
AnnotationTypesFolder	8.39694656	5.63380282	0
FontsColorsMIMEOptionFile	0	4.3956044	0
MIMEOptionFolder	0.37735849	3.7593985	0
OptionUtilities	4.07608696	-0.26109661	0
RADVisualContainer	-0.53475936	28.4946237	16.7364017
EventCustomEditor	-0.82987552	0	0
FormDataLoader	-36.4705882	-14.8148148	0
MethodPicker	-1.53846154	22.9166667	0
PersistenceManager	1.47058824	0	0
PropertyPicker	-1.58730159	22.0430108	0
RADContainer	0	0	0
AddAction	2.27272727	0	0
CustomizeLayoutAction	9.61538462	-5.26315789	0
EditContainerAction	13.3333333	-1.96078431	0
EditFormAction	13.3333333	-1.96078431	0
InPlaceEditAction	28.9473684	-4.08163265	0

NetBeans Class Stability Measurement			
Class Name	**CII (LOC)**		
	3.5.1/4.0	**4.0/5.0**	**5.0/5.5.1**
InstallBeanAction	0	13.6363636	0
InstallToPaletteAction	-43.3962264	10	0
ReloadAction	-8.82352941	0	0
TestAction	3.19148936	20.6185567	0
CustomCodeEditor	0	1.05263158	0
EnumEditor	7.27272727	0	0
StringArrayEditor	0	22.8346457	0
UnknownLayoutSupport	0	-18.1818182	0
GridLayoutSupport	-0.94339623	12.3809524	0
JToolBarSupport	0	20.8333333	0
NullLayoutSupport	-0.76335878	0	0
ScrollPaneSupport	15.8730159	0	0
PaletteItem	0	2.39520958	0
ScrollPopupMenu	16	5.17241379	0
SearchPanel	10.5527638	18.6363636	0
SearchTask	86.9565217	13.9534884	0
TextDetail	5.35714286	4.23728814	0
AntProjectDataLoader	-50	-35.0877193	0
ShortcutIterator	-33.1753555	0	0
EventSetPatternNode	-1.96078431	-14	0
IdxPropertyPatternNode	-2.54237288	-11.3043478	0
Pattern	0	32.4324324	0
PropertyActionSettings	-63.1578947	-7.14285714	0
PropertyPatternNode	-1.63043478	-5.52486188	0
ClassElementFinder	18.2320442	0	0
DocumentModelBuilder	-54.3307087	0	0
JavaModule	-15.5555556	-65.7894737	0
NodeFactoryPool	32.9896907	0	0
ParserAnnotation	14.5454545	0	0
ParserMessage	0	0	0
ContextDisplay	0	0	0
CookieDisplay	18.1102362	0	0
DispatchData	0	0	0
DisplayTable	10.9947644	0	0

NetBeans Class Stability Measurement			
Class Name	**CII (LOC)**		
	3.5.1/4.0	**4.0/5.0**	**5.0/5.5.1**
EditPanel	-9.27835052	-2.65151515	0
RequestData	0.69605568	0	0
TransactionNode	-5.11363636	38.9221557	0
BundleNodeCustomizer	-0.48076923	0	0
KeyNode	-2.30414747	0	0
LocaleNodeCustomizer	0	0	0
PresentableFileEntry	-8.48484848	0	0
PropertiesStructure	19.760479	0	0
PropertyBundleEvent	18.5185185	0	0
PropertyBundleSupport	10	0	0
UtilConvert	-62.2159091	0	0
AutoupdateType	7.04225352	2.63157895	3.84615385
CertificateDialog	-0.7751938	0	0
SetupPanel	1.61290323	0	0
Access	6.4516129	0	0
ClassName	8.95522388	0	0
Code	28.3783784	14.7368421	0
ConstantPoolReader	0	0	0
CPInterfaceMethodInfo	0	0	0
Field	47.4226804	0	0
Method	131.034483	0	0

TABLE B- 10 NetBeans Class Stability Measurement (Grosser)

NetBeans Class Stability Measurement			
Class Name	**Grosser(P-NOM)**		
	3.5.1/4.0	**4.0/5.0**	**5.0/5.5.1**
IDESettingsBeanInfo	1	1	1
NbPlaces	1	0.58	1
NonGuiMain	1	1	1
Plain	1	1	1
WarmUpSupport	1	1	1
AboutAction	1	0.8	1
ConfigureShortcutsAction	1	1	1
GlobalPropertiesAction	0.8	1	1
HTMLViewAction	1	1	1
SystemExit	0.8	1	1
ViewRuntimeTabAction	0.6	1	1
DefaultParser	0.94	1	1
XMLEnvironmentProvider	1	1	1
PerformanceEvent	1	1	1
FileStateEditor	1	1	1
XML	1	1	1
FormatterIndentEngine	1	1	1
NbEditorUtilities	0.89	1	1
AnnotationTypesFolder	1	1	1
FontsColorsMIMEOptionFile	1	1	1
MIMEOptionFolder	1	1	1
OptionUtilities	1	1	1
RADVisualContainer	1	0.88	1
EventCustomEditor	1	1	1
FormDataLoader	0.86	0.83	1
MethodPicker	1	1	1
PersistenceManager	1	1	1
PropertyPicker	1	1	1
RADContainer	1	1	1
AddAction	0.83	1	1
CustomizeLayoutAction	1	0.88	1
EditContainerAction	1	1	1
EditFormAction	1	1	1
InPlaceEditAction	1	1	1

NetBeans Class Stability Measurement			
Class Name	**Grosser (P-NOM)**		
	3.5.1/4.0	**4.0/5.0**	**5.0/5.5.1**
InstallBeanAction	0.75	1	1
InstallToPaletteAction	0.71	1	1
ReloadAction	0.67	1	1
TestAction	1	1	1
CustomCodeEditor	1	1	1
EnumEditor	1	1	1
StringArrayEditor	1	1	1
UnknownLayoutSupport	1	0.67	1
GridLayoutSupport	1	1	1
JToolBarSupport	1	1	1
NullLayoutSupport	1	1	1
ScrollPaneSupport	1	1	1
PaletteItem	0.33	1	1
ScrollPopupMenu	1	1	1
SearchPanel	1	1	1
SearchTask	0.67	1	1
TextDetail	1	1	1
AntProjectDataLoader	0.71	1	1
ShortcutIterator	0.64	1	1
EventSetPatternNode	0.92	0.83	1
IdxPropertyPatternNode	0.89	1	1
Pattern	0.88	1	1
PropertyActionSettings	0.37	0.86	1
PropertyPatternNode	0.92	0.92	1
ClassElementFinder	1	1	1
DocumentModelBuilder	0.95	1	1
JavaModule	1	0.67	1
NodeFactoryPool	1	1	1
ParserAnnotation	1	1	1
ParserMessage	1	1	1
ContextDisplay	1	1	1
CookieDisplay	1	1	1
DispatchData	1	1	1
DisplayTable	1	1	1

NetBeans Class Stability Measurement			
Class Name	**Grosser (P-NOM)**		
	3.5.1/4.0	**4.0/5.0**	**5.0/5.5.1**
EditPanel	0.17	1	1
RequestData	1	1	1
TransactionNode	0.94	1	1
BundleNodeCustomizer	1	1	1
KeyNode	0.93	1	1
LocaleNodeCustomizer	1	1	1
PresentableFileEntry	1	1	1
PropertiesStructure	0.9	1	1
PropertyBundleEvent	1	1	1
PropertyBundleSupport	1	1	1
UtilConvert	0.71	1	1
AutoupdateType	1	1	1
CertificateDialog	1	1	1
SetupPanel	1	1	1
Access	1	1	1
ClassName	1	1	1
Code	1	1	1
ConstantPoolReader	1	1	1
CPInterfaceMethodInfo	1	1	1
Field	1	1	1
Method	1	1	1

TABLE B- 11 NetBeans Class Stability Measurement (Stab)

NetBeans Class Stability Measurement			
Class Name	**Stab (NOM)**		
	3.5.1/4.0	**4.0/5.0**	**5.0/5.5.1**
IDESettingsBeanInfo	1	1	1
NbPlaces	1	0	1
NonGuiMain	1	1	1
Plain	1	1	1
WarmUpSupport	1	1	1
AboutAction	0	1	0
ConfigureShortcutsAction	0	1	1
GlobalPropertiesAction	1	1	1
HTMLViewAction	0	1	1
SystemExit	1	0	1
ViewRuntimeTabAction	0	1	1
DefaultParser	1	1	1
XMLEnvironmentProvider	1	1	1
PerformanceEvent	1	1	1
FileStateEditor	1	1	1
XML	1	1	1
FormatterIndentEngine	0	1	1
NbEditorUtilities	0	0	1
AnnotationTypesFolder	1	1	1
FontsColorsMIMEOptionFile	1	1	1
MIMEOptionFolder	1	1	1
OptionUtilities	0	1	1
RADVisualContainer	1	0	1
EventCustomEditor	1	1	1
FormDataLoader	0	1	1
MethodPicker	1	0	1
PersistenceManager	1	1	1
PropertyPicker	1	0	1
RADContainer	1	1	1
AddAction	0	1	1
CustomizeLayoutAction	0	0	1
EditContainerAction	0	1	1
EditFormAction	0	1	1
InPlaceEditAction	0	1	1

NetBeans Class Stability Measurement			
Class Name	**Stab (NOM)**		
	3.5.1/4.0	**4.0/5.0**	**5.0/5.5.1**
InstallBeanAction	1	0	1
InstallToPaletteAction	0	0	1
ReloadAction	0	1	1
TestAction	0	1	1
CustomCodeEditor	1	1	1
EnumEditor	1	1	1
StringArrayEditor	1	1	1
UnknownLayoutSupport	1	0	1
GridLayoutSupport	1	1	1
JToolBarSupport	1	0	1
NullLayoutSupport	1	1	1
ScrollPaneSupport	1	1	1
PaletteItem	0	1	1
ScrollPopupMenu	0	1	1
SearchPanel	0	0	1
SearchTask	0	0	1
TextDetail	0	1	1
AntProjectDataLoader	0	1	1
ShortcutIterator	0	1	1
EventSetPatternNode	0	0	1
IdxPropertyPatternNode	0	1	1
Pattern	1	1	1
PropertyActionSettings	0	0	1
PropertyPatternNode	0	0	1
ClassElementFinder	0	1	1
DocumentModelBuilder	0	1	1
JavaModule	1	0	1
NodeFactoryPool	0	1	1
ParserAnnotation	0	1	1
ParserMessage	1	1	1
ContextDisplay	1	1	1
CookieDisplay	1	1	1
DispatchData	1	1	1
DisplayTable	0	1	1

NetBeans Class Stability Measurement			
Class Name	**Stab (NOM)**		
	3.5.1/4.0	**4.0/5.0**	**5.0/5.5.1**
EditPanel	0	1	1
RequestData	1	1	1
TransactionNode	0	0	1
BundleNodeCustomizer	1	1	1
KeyNode	1	1	1
LocaleNodeCustomizer	1	1	1
PresentableFileEntry	0	1	1
PropertiesStructure	1	1	1
PropertyBundleEvent	1	1	1
PropertyBundleSupport	1	1	1
UtilConvert	0	1	1
AutoupdateType	1	1	1
CertificateDialog	1	1	1
SetupPanel	1	1	1
Access	1	1	1
ClassName	0	1	1
Code	0	0	1
ConstantPoolReader	1	1	1
CPInterfaceMethodInfo	1	1	1
Field	0	1	1
Method	0	1	1

TABLE B- 12 NetBeans Class Stability Measurement (CSM)

NetBeans Class Stability Measurement			
Class Name	**CSM**		
	3.5.1/4.0	**3.5.1/5.0**	**3.5.1/5.5.1**
IDESettingsBeanInfo	0.9375	0.78125	0.7291667
NbPlaces	0.9880952	0.6678571	0.6556122
NonGuiMain	0.9921875	0.80078125	0.74218756
Plain	0.796875	0.7395834	0.7109375
WarmUpSupport	0.9375	0.78125	0.7291667
AboutAction	0.65625	0.46875	0.4375
ConfigureShortcutsAction	1	0.796875	0.7395834
GlobalPropertiesAction	0.525	0.38750002	0.34166667
HTMLViewAction	0.6875	0.53125	0.47916666
SystemExit	0.77500004	0.63750005	0.5916667
ViewRuntimeTabAction	0.825	0.725	0.6916666
DefaultParser	0.9609375	0.79296875	0.7369792
XMLEnvironmentProvider	1	0.8125	0.75000006
PerformanceEvent	1	0.8125	0.75000006
FileStateEditor	0.95	0.7875	0.73333335
XML	1	0.8125	0.75000006
FormatterIndentEngine	0.9921875	0.80078125	0.74218756
NbEditorUtilities	1	0.80625	0.7458334
AnnotationTypesFolder	0.90625	0.8020833	0.671875
FontsColorsMIMEOptionFile	0.75000006	0.6770834	0.6666666
MIMEOptionFolder	0.975	0.78125	0.7291667
OptionUtilities	0.98	0.795	0.73833334
RADVisualContainer	0.9812499	0.753125	0.7083333
EventCustomEditor	0.984375	0.8046875	0.7447917
FormDataLoader	0.875	0.73214287	0.6964286
MethodPicker	1	0.7369506	0.680403
PersistenceManager	0.9875	0.80625	0.7458334
PropertyPicker	1	0.7480769	0.6903846
RADContainer	1	0.8125	0.75000006
AddAction	0.7291666	0.65277785	0.5572916
CustomizeLayoutAction	0.98214287	0.75892854	0.71428573
EditContainerAction	0.9375	0.765625	0.71875006
EditFormAction	0.9375	0.765625	0.71875006
InPlaceEditAction	0.9375	0.765625	0.71875006

NetBeans Class Stability Measurement			
Class Name	**CSM**		
	3.5.1/4.0	**3.5.1/5.0**	**3.5.1/5.5.1**
InstallBeanAction	0.875	0.75	0.7083333
InstallToPaletteAction	0.85714287	0.74107146	0.70238096
ReloadAction	0.5833334	0.46874997	0.43750003
TestAction	0.98214287	0.78571427	0.7321429
CustomCodeEditor	0.98214287	0.79464287	0.7380953
EnumEditor	0.995283	0.807783	0.74528307
StringArrayEditor	1	0.8055556	0.7453704
UnknownLayoutSupport	1	0.75000006	0.7083334
GridLayoutSupport	1	0.8125	0.75000006
JToolBarSupport	1	0.8125	0.75000006
NullLayoutSupport	1	0.8125	0.75000006
ScrollPaneSupport	0.75	0.5625	0.49999997
PaletteItem	0.328125	0.27604166	0.26953125
ScrollPopupMenu	0.9464286	0.7767857	0.72619045
SearchPanel	0.6333333	0.4875	0.44166666
SearchTask	0.53125	0.46875	0.4479167
TextDetail	0.9295455	0.74772733	0.6984849
AntProjectDataLoader	0.875	0.703125	0.6666667
ShortcutIterator	0.38777092	0.2857972	0.251806
EventSetPatternNode	0.9732142	0.625	0.5833333
IdxPropertyPatternNode	0.96250004	0.65624994	0.6041667
Pattern	1	0.7875	0.73333335
PropertyActionSettings	0.5899123	0.5109649	0.49342105
PropertyPatternNode	0.9765625	0.76953125	0.7213541
ClassElementFinder	0.815	0.71875	0.68749994
DocumentModelBuilder	0.9013158	0.7631579	0.7171052
JavaModule	0.9583333	0.4583333	0.43055558
NodeFactoryPool	0.7552083	0.6067708	0.5572917
ParserAnnotation	0.9553571	0.7901786	0.735119
ParserMessage	1	0.8125	0.75000006
ContextDisplay	1	0.8125	0.75000006
CookieDisplay	0.815	0.65625	0.6041667
DispatchData	1	0.8125	0.75000006
DisplayTable	0.9947917	0.8098958	0.74826396

NetBeans Class Stability Measurement			
Class Name	**CSM**		
	3.5.1/4.0	**3.5.1/5.0**	**3.5.1/5.5.1**
EditPanel	0.6355219	0.54461277	0.51430976
RequestData	0.80668604	0.7461241	0.715843
TransactionNode	0.9671053	0.7730263	0.7236842
BundleNodeCustomizer	1	0.8125	0.75000006
KeyNode	0.9485294	0.73284316	0.70588243
LocaleNodeCustomizer	0.97115386	0.7980769	0.74038464
PresentableFileEntry	0.97115386	0.7980769	0.74038464
PropertiesStructure	0.875	0.75	0.7083333
PropertyBundleEvent	0.98214287	0.8035714	0.74404764
PropertyBundleSupport	0.9375	0.78125	0.7291667
UtilConvert	0.8333333	0.7291667	0.6944445
AutoupdateType	0.9895833	0.8020833	0.7395834
CertificateDialog	0.85	0.6625	0.60833335
SetupPanel	0.8541667	0.6770833	0.6180556
Access	0.98214287	0.8035714	0.74404764
ClassName	0.9666667	0.79583335	0.7388889
Code	0.9375	0.78125	0.7291667
ConstantPoolReader	1	0.8125	0.75000006
CPInterfaceMethodInfo	1	0.8125	0.75000006
Field	0.6328125	0.5885417	0.56640625
Method	0.7916667	0.6666666	0.625

Appendix C

The Selected Software Metrics

TABLE C- 1 Android Software Metrics V1.5

Android 1.5										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
Typeface	2	2	11	13	19	13	0	2	58	66
Scroller	6	2	27	19	38	24	0	1	3	195
DeleteEventHelper	6	2	19	4	27	4	1	2	0	190
OnScreenHint	8	2	15	17	44	17	2	3	72	157
EmailAddressValidator	1	2	0	2	13	2	0	1	1	13
MessagingListener	1	2	0	28	39	28	0	2	378	84
AlarmManager	1	2	10	5	16	5	0	1	0	53
NotificationManager	3	2	5	4	20	4	1	1	0	62
StatusBarManager	1	2	7	7	18	7	0	1	0	69
AppWidgetProvider	9	2	0	5	24	5	0	1	10	44
ActivityNotFoundException	1	5	0	0	21	0	0	1	0	11
ContentUris	1	2	0	3	14	4	0	2	3	14
MutableContextWrapper	1	2	0	1	12	1	0	0	0	9
ReceiverCallNotAllowedException	1	2	0	0	11	0	0	1	0	7
SyncContext	2	2	3	4	16	4	0	1	0	33
TypedArray	6	2	7	30	90	36	0	8	0	372
CharArrayBuffer	1	2	2	0	11	0	0	0	0	11
CursorIndexOutOfBoundsException	1	6	0	0	21	0	0	1	0	9
DataSetObserver	1	2	0	2	13	2	0	0	1	7

Android 1.5										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
SQLiteDiskIOException	1	3	0	0	11	0	0	1	0	7
SQLiteDoneException	1	3	0	0	11	0	0	1	0	7
SQLiteException	1	2	0	0	11	0	0	1	0	8
SQLiteFullException	1	3	0	0	11	0	0	1	0	7
SQLiteMisuseException	1	3	0	0	11	0	0	1	0	7
SQLiteOpenHelper	11	2	7	6	32	7	1	4	9	110
SQLiteStatement	3	2	2	7	23	7	1	1	9	94
GpsSatellite	1	2	8	7	19	8	0	0	16	44
LocationProvider	6	2	6	11	18	12	1	1	55	29
Ringtone	9	2	10	8	40	12	2	5	0	149
FocusFinderHelper	1	2	1	6	17	6	0	0	3	26
Gravity	15	2	23	5	52	5	0	0	10	159
InflateException	1	5	0	0	21	0	0	1	0	15
SoundEffectConstants	1	2	5	1	18	1	0	0	0	25
TouchDelegate	6	2	9	1	21	1	0	0	0	64
WindowManagerImpl	9	2	17	14	54	17	0	3	54	249
AccelerateInterpolator	2	2	1	1	13	2	0	1	0	25
BaseInputConnection	22	2	9	25	107	25	1	2	142	390
DocumentBuilder	2	2	0	13	21	14	0	11	78	57

Android 1.5										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
ClassFormatError	1	5	1	0	21	0	0	1	0	10
Error	1	3	1	0	21	0	0	1	0	16
Exception	1	3	1	0	21	0	0	1	0	16
IllegalAccessError	1	6	1	0	21	0	0	1	0	10
IllegalAccessException	1	4	1	0	21	0	0	1	0	10
IllegalStateException	1	5	1	0	21	0	0	1	0	16
Math	5	2	3	46	56	56	1	4	1033	157
Number	1	2	1	6	13	6	0	0	15	16
OutOfMemoryError	1	5	1	0	21	0	0	1	0	10
Process	0	2	0	6	11	6	0	3	15	11
RuntimeException	1	4	1	0	21	0	0	1	0	16
RuntimePermission	1	4	16	0	18	0	0	1	0	41
SecurityException	1	5	1	0	21	0	0	1	0	16
StackOverflowError	1	5	1	0	21	0	0	1	0	10
StrictMath	5	2	3	46	62	58	1	4	1033	175
StringIndexOutOfBoundsException	1	6	1	0	21	0	0	1	0	14
ThreadDeath	1	4	1	0	21	0	0	0	0	6
TypeNotPresentException	1	5	2	1	22	1	1	1	0	12
UnknownError	1	5	1	0	21	0	0	1	0	10

Android 1.5										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
SQLiteAbortException	1	3	0	0	11	0	0	1	0	7
SQLiteClosable	2	2	2	5	18	5	0	0	4	32
SQLiteConstraintException	1	3	0	0	11	0	0	1	0	7
SQLiteDatabaseCorruptException	1	3	0	0	11	0	0	1	0	7
SAXParser	3	2	0	18	38	31	0	13	153	129
AbstractMethodError	1	6	1	0	21	0	0	1	0	10
ArrayStoreException	1	5	1	0	21	0	0	1	0	10
AssertionError	1	4	1	0	21	7	0	0	0	30
Vibrator	2	2	1	3	15	3	0	0	0	35
AuthProvider	1	2	1	3	11	3	0	4	3	13
CollationElementIterator	1	2	2	11	22	11	0	1	0	43
Environment	1	2	13	5	17	8	2	4	15	40
SystemProperties	6	2	2	9	26	14	0	4	6	66
SQLException	1	5	0	0	21	0	0	1	0	9
StaleDataException	1	5	0	0	21	0	0	1	0	12
FactoryConfigurationError	3	4	1	2	24	2	1	2	0	31
ParserConfigurationException	1	4	0	0	21	0	0	1	0	9
VerifyError	1	5	1	0	21	0	0	1	0	10
AlgorithmParameterGenerator	3	2	6	10	26	11	2	3	23	82

TABLE C- 2 Android Software Metrics V1.6

Android 1.6										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
Typeface	2	2	11	16	21	16	0	3	100	73
Scroller	6	2	27	19	38	24	0	1	3	195
DeleteEventHelper	6	2	19	4	27	4	1	2	0	190
OnScreenHint	5	2	15	10	30	10	2	3	11	126
EmailAddressValidator	1	2	0	2	13	2	0	1	1	13
MessagingListener	1	2	0	34	45	34	0	2	561	96
AlarmManager	1	2	10	5	16	5	0	1	0	53
NotificationManager	3	2	5	4	20	4	1	1	0	62
StatusBarManager	1	2	7	7	18	7	0	1	0	69
AppWidgetProvider	8	2	0	5	23	5	0	1	10	42
ActivityNotFoundException	1	5	0	0	21	0	0	1	0	11
ContentUris	1	2	0	3	14	4	0	2	3	14
MutableContextWrapper	1	2	0	1	12	1	0	0	0	9
ReceiverCallNotAllowedException	1	2	0	0	11	0	0	1	0	7
SyncContext	2	2	3	4	16	4	0	1	0	33
TypedArray	6	2	7	30	90	36	0	8	0	373
CharArrayBuffer	1	2	2	0	11	0	0	0	0	11
CursorIndexOutOfBoundsException	1	6	0	0	21	0	0	1	0	9
DataSetObserver	1	2	0	2	13	2	0	0	1	7
SQLException	1	5	0	0	21	0	0	1	0	9

Android 1.6										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
SQLiteAbortException	1	3	0	0	11	0	0	1	0	7
SQLiteClosable	2	2	2	5	18	5	0	0	4	32
SQLiteConstraintException	1	3	0	0	11	0	0	1	0	7
SQLiteDatabaseCorruptException	1	3	0	0	11	0	0	1	0	7
SQLiteDiskIOException	1	3	0	0	11	0	0	1	0	7
SQLiteDoneException	1	3	0	0	11	0	0	1	0	7
SQLiteException	1	2	0	0	11	0	0	1	0	8
SQLiteFullException	1	3	0	0	11	0	0	1	0	7
SQLiteMisuseException	1	3	0	0	11	0	0	1	0	7
SQLiteOpenHelper	11	2	7	6	32	7	1	4	9	110
SQLiteStatement	3	2	2	7	23	7	1	1	9	94
GpsSatellite	1	2	8	7	19	8	0	0	16	44
LocationProvider	6	2	6	11	18	12	1	1	55	49
Ringtone	9	2	10	8	40	12	2	5	0	149
FocusFinderHelper	1	2	1	6	17	6	0	0	3	26
Gravity	15	2	23	5	52	5	0	0	10	159
InflateException	1	5	0	0	21	0	0	1	0	15
SoundEffectConstants	1	2	5	1	18	1	0	0	0	25
TouchDelegate	6	2	9	1	21	1	0	0	0	64
WindowManagerImpl	9	2	17	14	54	17	0	3	54	249
AccelerateInterpolator	2	2	2	1	13	2	0	1	0	30

Android 1.6										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
BaseInputConnection	23	2	9	25	109	25	1	2	142	402
DocumentBuilder	2	2	0	13	21	14	0	11	78	57
FactoryConfigurationError	3	4	1	2	24	2	1	2	0	31
ParserConfigurationException	1	4	0	0	21	0	0	1	0	9
SAXParser	3	2	0	18	38	31	0	13	153	129
AbstractMethodError	1	6	1	0	21	0	0	1	0	10
ArrayStoreException	1	5	1	0	21	0	0	1	0	10
AssertionError	1	4	1	0	21	7	0	0	0	30
ClassFormatError	1	5	1	0	21	0	0	1	0	10
Error	1	3	1	0	21	0	0	1	0	16
Exception	1	3	1	0	21	0	0	1	0	16
IllegalAccessError	1	6	1	0	21	0	0	1	0	10
IllegalAccessException	1	4	1	0	21	0	0	1	0	10
IllegalStateException	1	5	1	0	21	0	0	1	0	16
Math	5	2	3	46	56	56	1	4	1033	157
Number	1	2	1	6	13	6	0	0	15	16
OutOfMemoryError	1	5	1	0	21	0	0	1	0	10
Process	0	2	0	6	11	6	0	3	15	11
RuntimeException	1	4	1	0	21	0	0	1	0	16
RuntimePermission	1	4	16	0	18	0	0	1	0	41
SecurityException	1	5	1	0	21	0	0	1	0	16

Android 1.6										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
StackOverflowError	1	5	1	0	21	0	0	1	0	10
StrictMath	5	2	3	46	62	58	1	4	1033	175
StringIndexOutOfBoundsException	1	6	1	0	21	0	0	1	0	14
ThreadDeath	1	4	1	0	21	0	0	0	0	6
TypeNotPresentException	1	5	2	1	22	1	1	1	0	12
UnknownError	1	5	1	0	21	0	0	1	0	10
VerifyError	1	5	1	0	21	0	0	1	0	10
AlgorithmParameterGenerator	3	2	6	10	26	11	2	3	23	82
AuthProvider	1	2	1	3	11	3	0	4	3	13
CollationElementIterator	1	2	2	11	22	11	0	1	0	43
Environment	1	2	13	5	17	8	2	4	15	40
SystemProperties	6	2	2	9	26	14	0	4	6	66
Vibrator	2	2	1	3	15	3	0	0	0	35
StaleDataException	1	5	0	0	21	0	0	1	0	12

TABLE C- 3 Android Software Metrics V2.1

Android 2.1										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
Typeface	2	2	11	17	21	17	0	3	116	74
Scroller	6	2	27	20	39	25	0	1	10	199
DeleteEventHelper	6	2	18	4	27	4	1	2	0	189
OnScreenHint	5	2	15	6	26	6	2	3	0	112
EmailAddressValidator	1	2	0	2	13	2	0	1	1	13
MessagingListener	1	2	0	19	30	19	0	2	171	58
AlarmManager	1	2	10	5	16	5	0	1	0	53
NotificationManager	3	2	5	6	22	6	1	1	0	70
StatusBarManager	1	2	7	7	18	7	0	1	0	69
AppWidgetProvider	8	2	0	5	23	5	0	1	10	42
ActivityNotFoundException	1	5	0	0	21	0	0	1	0	11
ContentUris	1	2	0	3	14	4	0	2	3	14
MutableContextWrapper	1	2	0	1	12	1	0	0	0	9
ReceiverCallNotAllowedException	1	2	0	0	11	0	0	1	0	7
SyncContext	2	2	3	4	16	4	0	1	0	34
TypedArray	6	2	7	30	90	36	0	8	0	373
CharArrayBuffer	1	2	2	0	11	0	0	0	0	11
CursorIndexOutOfBoundsException	1	6	0	0	21	0	0	1	0	9
DataSetObserver	1	2	0	2	13	2	0	0	1	7
SQLException	1	5	0	0	21	0	0	1	0	9

Android 2.1										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
StaleDataException	1	5	0	0	21	0	0	1	0	12
SQLiteAbortException	1	3	0	0	11	0	0	1	0	7
SQLiteClosable	2	2	2	5	18	5	0	0	4	32
SQLiteConstraintException	1	3	0	0	11	0	0	1	0	7
SQLiteDatabaseCorruptException	1	3	0	0	11	0	0	1	0	7
SQLiteDiskIOException	1	3	0	0	11	0	0	1	0	7
SQLiteDoneException	1	3	0	0	11	0	0	1	0	7
SQLiteException	1	2	0	0	11	0	0	1	0	8
SQLiteFullException	1	3	0	0	11	0	0	1	0	7
SQLiteMisuseException	1	3	0	0	11	0	0	1	0	7
SQLiteOpenHelper	11	2	7	6	32	7	1	4	9	110
SQLiteStatement	3	2	2	7	23	7	1	1	9	94
GpsSatellite	1	2	8	7	19	8	0	0	16	44
LocationProvider	6	2	6	11	18	12	1	1	55	49
Ringtone	9	2	10	8	40	12	2	5	0	149
FocusFinderHelper	1	2	1	6	17	6	0	0	3	26
Gravity	15	2	23	5	52	5	0	0	10	159
InflateException	1	5	0	0	21	0	0	1	0	15
SoundEffectConstants	1	2	5	1	18	1	0	0	0	25
TouchDelegate	6	2	9	1	21	1	0	0	0	64

Android 2.1										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
WindowManagerImpl	9	2	17	14	54	17	0	3	54	249
AccelerateInterpolator	2	2	2	1	13	2	0	1	0	30
BaseInputConnection	23	2	9	25	109	25	1	2	142	402
DocumentBuilder	2	2	0	13	21	14	0	11	78	57
FactoryConfigurationError	3	4	1	2	24	2	1	2	0	31
ParserConfigurationException	1	4	0	0	21	0	0	1	0	9
SAXParser	3	2	0	18	38	31	0	13	153	129
AbstractMethodError	1	6	1	0	21	0	0	1	0	10
ArrayStoreException	1	5	1	0	21	0	0	1	0	10
AssertionError	1	4	1	0	21	7	0	0	0	30
ClassFormatError	1	5	1	0	21	0	0	1	0	10
Error	1	3	1	0	21	0	0	1	0	16
Exception	1	3	1	0	21	0	0	1	0	16
IllegalAccessError	1	6	1	0	21	0	0	1	0	10
IllegalAccessException	1	4	1	0	21	0	0	1	0	10
IllegalStateException	1	5	1	0	21	0	0	1	0	16
Math	5	2	3	46	56	58	1	4	1033	171
Number	1	2	1	6	13	6	0	0	15	16
OutOfMemoryError	1	5	1	0	21	0	0	1	0	10
Process	0	2	0	6	11	6	0	3	15	11

Android 2.1										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
RuntimeException	1	4	1	0	21	0	0	1	0	16
RuntimePermission	1	4	16	0	18	0	0	1	0	41
SecurityException	1	5	1	0	21	0	0	1	0	16
StackOverflowError	1	5	1	0	21	0	0	1	0	10
StrictMath	5	2	3	46	62	58	1	4	1033	175
StringIndexOutOfBoundsException	1	6	1	0	21	0	0	1	0	14
ThreadDeath	1	4	1	0	21	0	0	0	0	6
TypeNotPresentException	1	5	2	1	22	1	1	1	0	12
UnknownError	1	5	1	0	21	0	0	1	0	10
VerifyError	1	5	1	0	21	0	0	1	0	10
AlgorithmParameterGenerator	3	2	6	10	26	11	2	3	23	82
AuthProvider	1	2	1	3	11	3	0	4	3	13
CollationElementIterator	1	2	2	11	22	11	0	1	0	43
Environment	1	2	13	5	17	8	2	4	15	40
SystemProperties	3	2	2	12	24	13	0	1	36	52
Vibrator	2	2	2	3	15	3	0	0	0	36

TABLE C- 4 Android Software Metrics V2.3.1

Android 2.3.1										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
Typeface	2	2	11	17	21	17	0	3	116	77
Scroller	5	2	27	19	38	25	0	1	14	198
DeleteEventHelper	6	2	18	4	27	4	1	2	0	190
OnScreenHint	5	2	14	6	26	6	2	3	0	111
EmailAddressValidator	1	2	0	2	13	2	0	1	1	11
MessagingListener	1	2	0	19	30	19	0	2	171	56
AlarmManager	1	2	10	6	17	6	0	1	0	59
NotificationManager	3	2	5	6	22	6	1	1	0	71
StatusBarManager	1	2	8	6	17	6	0	1	0	64
AppWidgetProvider	8	2	0	5	23	5	0	1	10	42
ActivityNotFoundException	1	5	0	0	21	0	0	1	0	11
ContentUris	1	2	0	3	14	4	0	2	3	14
MutableContextWrapper	1	2	0	1	12	1	0	0	0	9
ReceiverCallNotAllowedException	1	2	0	0	11	0	0	1	0	7
SyncContext	3	2	3	4	18	4	0	1	0	38
TypedArray	6	2	7	31	94	37	0	8	0	393
CharArrayBuffer	1	2	2	0	11	0	0	0	0	11
CursorIndexOutOfBoundsException	1	6	0	0	21	0	0	1	0	9
DataSetObserver	1	2	0	2	13	2	0	0	1	7
SQLException	1	5	0	0	21	0	0	1	0	9

Android 2.3.1										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
StaleDataException	1	5	0	0	21	0	0	1	0	12
SQLiteAbortException	1	3	0	0	11	0	0	1	0	7
SQLiteClosable	4	2	2	6	22	8	0	3	9	51
SQLiteConstraintException	1	3	0	0	11	0	0	1	0	7
SQLiteDatabaseCorruptException	1	3	0	0	11	0	0	1	0	7
SQLiteDiskIOException	1	3	0	0	11	0	0	1	0	7
SQLiteDoneException	1	3	0	0	11	0	0	1	0	7
SQLiteException	1	2	0	0	11	0	0	1	0	8
SQLiteFullException	1	3	0	0	11	0	0	1	0	7
SQLiteMisuseException	1	3	0	0	11	0	0	1	0	7
SQLiteOpenHelper	12	2	7	6	33	7	1	4	9	114
SQLiteStatement	2	2	0	7	19	7	0	1	21	79
GpsSatellite	1	2	8	7	19	8	0	0	16	44
LocationProvider	2	2	7	11	13	12	1	1	53	40
Ringtone	10	2	11	8	42	12	2	5	0	159
FocusFinderHelper	1	2	1	6	17	6	0	0	3	26
Gravity	15	2	23	5	52	5	0	0	10	159
InflateException	1	5	0	0	21	0	0	1	0	15
SoundEffectConstants	1	2	5	1	18	1	0	0	0	25
TouchDelegate	6	2	9	1	21	1	0	0	0	64

Android 2.3.1										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
WindowManagerImpl	9	2	17	14	54	17	0	3	54	249
AccelerateInterpolator	2	2	2	1	13	2	0	1	0	30
BaseInputConnection	18	2	9	29	126	29	1	2	174	458
DocumentBuilder	3	2	1	14	23	20	0	15	91	85
FactoryConfigurationError	2	4	1	2	23	2	1	2	0	30
ParserConfigurationException	1	4	0	0	21	0	0	1	0	9
SAXParser	3	2	1	19	38	35	0	17	169	159
AbstractMethodError	1	6	1	0	21	0	0	1	0	10
ArrayStoreException	1	5	1	0	21	0	0	1	0	10
AssertionError	1	4	1	0	21	7	0	0	0	30
ClassFormatError	1	5	1	0	21	0	0	1	0	10
Error	1	3	1	0	21	0	0	1	0	16
Exception	1	3	1	0	21	0	0	1	0	16
IllegalAccessError	1	6	1	0	21	0	0	1	0	10
IllegalAccessException	1	4	1	0	21	0	0	1	0	10
IllegalStateException	1	5	1	0	21	0	0	1	0	16
Math	12	2	3	58	118	71	1	5	1651	405
Number	1	2	1	6	13	6	0	0	15	16
OutOfMemoryError	1	5	1	0	21	0	0	1	0	10
Process	0	2	0	6	11	6	0	3	15	11

Android 2.3.1										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
RuntimeException	1	4	1	0	21	0	0	1	0	16
RuntimePermission	1	4	16	0	18	0	0	1	0	41
SecurityException	1	5	1	0	21	0	0	1	0	16
StackOverflowError	1	5	1	0	21	0	0	1	0	10
StrictMath	8	2	3	58	86	90	1	6	1653	283
StringIndexOutOfBoundsException	1	6	1	0	21	0	0	1	0	13
ThreadDeath	1	4	1	0	21	0	0	0	0	6
TypeNotPresentException	1	5	2	1	22	1	1	1	0	12
UnknownError	1	5	1	0	21	0	0	1	0	10
VerifyError	1	5	1	0	21	0	0	1	0	10
AlgorithmParameterGenerator	3	2	6	10	26	11	1	2	23	80
AuthProvider	1	2	1	3	11	3	0	4	3	13
CollationElementIterator	1	2	2	11	22	11	0	1	0	43
Environment	2	2	27	15	30	18	2	5	92	110
SystemProperties	3	2	2	12	24	13	0	1	36	52
Vibrator	3	2	3	3	18	3	1	1	0	52

TABLE C- 5 Eclipse Software Metrics V2.0

Eclipse 2.0										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
JDTCompilerAdapter	12	2	1	3	29	9	1	8	3	108
ClasspathDirectory	10	2	7	4	34	19	2	6	0	93
FileFinder	8	2	3	2	21	10	1	5	0	36
CodeFormatter	65	2	25	47	471	67	1	14	347	1872
AbstractMethodDeclaration	10	2	16	23	78	25	1	7	99	297
AbstractVariableDeclaration	3	2	8	2	14	2	0	1	1	24
Argument	4	2	0	4	25	4	0	1	6	62
ArrayAllocationExpression	14	2	4	5	44	6	1	5	0	126
ArrayInitializer	15	2	2	5	54	5	0	3	0	154
ClassFileReader	37	3	17	36	183	59	3	17	292	543
ClassFileStruct	15	2	2	14	53	21	0	7	0	161
ClassFormatException	1	4	31	1	22	1	0	0	0	44
FieldInfo	5	3	10	14	102	19	0	4	0	233
InnerClassInfo	4	3	11	6	68	10	0	3	0	89
MethodInfo	6	3	11	14	82	20	0	4	0	161
ConditionalFlowInfo	1	3	2	19	33	19	0	1	0	82
ExceptionHandlingFlowContext	5	3	9	6	114	12	1	5	0	142
FlowInfo	2	2	1	23	17	23	0	1	251	48
InitializationFlowContext	2	4	4	2	118	3	1	3	0	64
LabelFlowContext	3	4	1	2	99	4	0	1	0	35

Eclipse 2.0										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
LoopingFlowContext	8	4	6	6	116	11	0	2	20	121
BooleanConstant	2	3	1	4	937	5	0	2	0	22
ByteConstant	2	3	1	10	937	11	0	2	0	40
CompilerOptions	87	2	67	5	52	14	1	8	2	522
Constant	5	2	4	42	936	44	0	3	519	1254
IntConstant	2	3	1	10	937	11	0	2	0	41
LongConstant	2	3	1	10	937	11	0	2	0	40
StringConstant	2	3	1	4	938	5	1	1	0	21
RecoveredBlock	6	4	5	16	110	20	1	8	8	203
RecoveredField	5	3	4	10	83	14	1	6	0	117
RecoveredImport	2	3	1	6	69	6	0	1	0	31
RecoveredInitializer	7	4	3	10	102	14	2	7	37	172
RecoveredMethod	20	3	5	15	132	21	2	7	0	316
BufferedContent	3	2	2	8	25	8	0	1	4	53
CompareConfiguration	4	2	19	25	45	27	2	2	270	174
CompareUI	2	2	8	12	25	12	1	3	66	66
CompareViewerPane	2	2	1	5	19	5	0	1	8	57
CompareViewerSwitchingPane	11	3	9	20	64	22	1	3	132	200
HistoryItem	1	2	2	5	16	5	0	2	2	30
NavigationAction	3	2	2	2	15	2	0	1	0	27

Eclipse 2.0										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
ResourceNode	5	3	2	13	45	17	1	4	0	92
AddFromHistoryAction	7	2	2	4	23	7	1	5	4	100
BinaryCompareViewer	14	2	6	4	29	8	2	7	2	94
ChangePropertyAction	2	2	4	3	15	3	2	2	0	31
CompareAction	2	2	1	2	15	2	0	0	0	24
CompareMessages	1	2	2	1	12	3	2	3	0	16
DocLineComparator	7	2	5	7	28	11	0	2	5	111
DocumentManager	6	2	3	4	26	9	1	3	0	42
ExceptionHandler	3	2	1	8	24	12	0	5	26	62
ImageMergeViewer	7	2	7	7	26	9	1	4	5	101
ResizableDialog	10	2	8	4	27	4	2	3	0	112
SimpleTextViewer	3	2	2	4	18	4	0	1	2	44
DiffContainer	4	3	2	6	31	7	1	2	0	51
DiffElement	2	2	2	6	17	6	0	1	11	32
DiffNode	9	4	6	18	74	22	0	3	83	175
DocumentRangeNode	7	2	7	18	43	26	2	3	65	175
DebugEvent	3	3	18	5	34	7	0	2	0	114
DebugException	1	2	5	0	11	0	0	0	0	13
Launch	9	2	7	24	62	34	3	6	100	194
Context	2	2	3	8	20	9	1	3	22	45

Eclipse 2.0										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
HrefUtil	6	2	0	4	30	14	0	1	6	54
Link	2	2	2	3	15	7	2	5	3	25
Toc	8	2	9	10	31	21	4	11	39	110
TocFile	2	2	7	12	25	13	3	5	50	74
TocFileParser	6	3	3	6	36	20	2	10	9	92
TocManager	6	2	2	8	39	18	2	9	8	136
URLCoder	3	2	0	4	21	10	0	6	6	49
OverlayIcon	6	2	5	6	34	6	0	0	3	95
PDERuntimePlugin	2	2	2	10	23	13	1	3	23	67
PDERuntimePluginImages	1	2	40	5	16	8	2	4	0	107
RegistryBrowserLabelProvider	9	2	15	3	38	4	1	3	1	129
Color	4	2	2	10	32	13	1	2	0	72
Cursor	3	2	2	3	14	4	1	2	1	24
Device	6	2	46	19	71	32	4	10	121	252
Font	5	2	2	7	25	11	1	4	0	62
FontData	3	2	8	11	27	21	1	4	7	89
FontMetrics	3	2	5	8	19	8	0	0	0	41
Region	4	2	1	14	38	14	0	1	0	76
Canvas	6	2	1	6	46	16	1	1	0	98
Caret	7	2	11	18	77	32	1	2	0	221
ColorDialog	3	2	1	3	16	3	0	0	0	39

Eclipse 2.0										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
DirectoryDialog	7	2	2	5	25	6	1	3	2	64
FileDialog	11	2	5	10	38	12	1	4	37	123
Group	3	2	2	5	26	11	0	3	45	64
Label	9	2	4	11	82	27	1	3	184	249
MessageBox	8	2	1	3	26	5	1	3	0	59
ProgressBar	4	2	0	7	32	13	0	0	78	78
Sash	5	2	6	3	38	12	0	0	60	150
Scale	4	2	2	13	44	17	0	0	132	119
Scrollable	11	2	5	5	54	18	0	2	76	185
TabItem	8	2	3	9	36	11	1	1	5	89
ToolBar	5	2	4	12	76	28	0	3	49	237

TABLE C- 6 Eclipse Software Metrics V2.0.2

Eclipse 2.0.2										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
JDTCompilerAdapter	13	2	1	3	30	12	1	11	3	117
ClasspathDirectory	10	2	7	4	34	19	2	6	0	93
FileFinder	8	2	3	2	21	10	1	5	0	36
CodeFormatter	65	2	25	47	471	67	1	14	347	1872
AbstractMethodDeclaration	10	2	16	23	78	25	1	7	99	297
AbstractVariableDeclaration	3	2	8	2	14	2	0	1	1	24
Argument	4	2	0	4	25	4	0	1	6	62
ArrayAllocationExpression	14	2	4	5	44	6	1	5	0	126
ArrayInitializer	15	2	2	5	54	5	0	3	0	154
ClassFileReader	37	3	17	36	183	59	3	17	292	543
ClassFileStruct	15	2	2	14	53	21	0	7	0	161
ClassFormatException	1	4	31	1	22	1	0	0	0	44
FieldInfo	5	3	10	14	102	19	0	4	0	233
InnerClassInfo	4	3	11	6	68	10	0	3	0	89
MethodInfo	6	3	11	14	82	20	0	4	0	161
ConditionalFlowInfo	1	3	2	19	33	19	0	1	0	82
ExceptionHandlingFlowContext	5	3	9	6	114	12	1	5	0	142
FlowInfo	2	2	1	23	17	23	0	1	251	48
InitializationFlowContext	2	4	4	2	118	3	1	3	0	64
LabelFlowContext	3	4	1	2	99	4	0	1	0	35

Eclipse 2.0.2										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
LoopingFlowContext	8	4	6	6	116	11	0	2	20	121
BooleanConstant	2	3	1	4	937	5	0	2	0	22
ByteConstant	2	3	1	10	937	11	0	2	0	40
CompilerOptions	87	2	67	5	52	14	1	8	2	522
Constant	5	2	4	42	936	44	0	3	519	1254
IntConstant	2	3	1	10	937	11	0	2	0	41
LongConstant	2	3	1	10	937	11	0	2	0	40
StringConstant	2	3	1	4	938	5	1	1	0	21
RecoveredBlock	6	4	5	16	110	20	1	8	8	203
RecoveredField	5	3	4	10	83	14	1	6	0	117
RecoveredImport	2	3	1	6	69	6	0	1	0	31
RecoveredInitializer	7	4	3	10	102	14	2	7	37	172
RecoveredMethod	20	3	5	15	132	21	2	7	0	316
BufferedContent	3	2	2	8	25	8	0	1	4	53
CompareConfiguration	4	2	19	25	45	27	2	2	270	174
CompareUI	2	2	8	12	25	12	1	3	66	66
CompareViewerPane	2	2	1	5	19	5	0	1	8	57
CompareViewerSwitchingPane	11	3	9	20	64	22	1	3	132	200
HistoryItem	1	2	2	5	16	5	0	2	2	30
NavigationAction	3	2	2	2	15	2	0	1	0	27

Eclipse 2.0.2										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
ResourceNode	5	3	2	13	45	17	1	4	0	92
AddFromHistoryAction	7	2	2	4	23	7	1	5	4	100
BinaryCompareViewer	14	2	6	4	29	8	2	7	2	94
ChangePropertyAction	2	2	4	3	15	3	2	2	0	31
CompareAction	2	2	1	2	15	2	0	0	0	24
CompareMessages	1	2	2	1	12	3	2	3	0	16
DocLineComparator	7	2	5	7	28	11	0	2	5	111
DocumentManager	6	2	3	4	26	9	1	3	0	42
ExceptionHandler	3	2	1	8	24	12	0	5	26	62
ImageMergeViewer	7	2	7	7	26	9	1	4	5	101
ResizableDialog	10	2	8	4	27	4	2	3	0	112
SimpleTextViewer	3	2	2	4	18	4	0	1	2	44
DiffContainer	4	3	2	6	31	7	1	2	0	51
DiffElement	2	2	2	6	17	6	0	1	11	32
DiffNode	9	4	6	18	74	22	0	3	83	175
DocumentRangeNode	7	2	7	18	43	26	2	3	65	175
DebugEvent	3	3	18	5	34	7	0	2	0	114
DebugException	1	2	5	0	11	0	0	0	0	13
Launch	9	2	7	24	62	34	3	6	100	194
Context	2	2	3	8	20	9	1	3	22	45

Eclipse 2.0.2										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
HrefUtil	6	2	0	4	30	14	0	1	6	54
Link	2	2	2	3	15	7	2	5	3	25
Toc	8	2	9	10	31	21	4	11	39	110
TocFile	2	2	7	12	25	13	3	5	50	74
TocFileParser	6	3	3	6	36	20	2	10	9	92
TocManager	6	2	2	8	39	18	2	9	8	136
URLCoder	3	2	0	4	21	10	0	6	6	49
OverlayIcon	6	2	5	6	34	6	0	0	3	95
PDERuntimePlugin	2	2	2	10	23	13	1	3	23	67
PDERuntimePluginImages	1	2	40	5	16	8	2	4	0	107
RegistryBrowserLabelProvider	9	2	15	3	38	4	1	3	1	129
Color	4	2	2	10	32	13	1	2	0	72
Cursor	3	2	2	3	14	4	1	2	1	24
Device	6	2	46	19	71	32	4	10	121	252
Font	5	2	2	7	25	11	1	4	0	62
FontData	3	2	8	11	24	21	1	4	7	89
FontMetrics	3	2	5	8	19	8	0	0	0	41
Region	4	2	1	14	38	14	0	1	0	76
Canvas	6	2	1	6	46	16	1	1	0	98
Caret	7	2	11	18	77	32	1	2	0	221

Eclipse 2.0.2										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
ColorDialog	3	2	1	3	16	3	0	0	0	39
DirectoryDialog	7	2	2	5	25	6	1	3	2	64
FileDialog	11	2	5	10	38	12	1	4	37	123
Group	3	2	2	5	26	11	0	3	45	64
Label	9	2	4	11	82	27	1	3	184	249
MessageBox	8	2	1	3	26	5	1	3	0	59
ProgressBar	4	2	0	7	32	13	0	0	78	78
Sash	5	2	6	3	38	12	0	0	60	150
Scale	4	2	2	13	44	17	0	0	132	119
Scrollable	11	2	5	5	54	18	0	2	76	185
TabItem	8	2	3	9	36	11	1	1	5	89
ToolBar	5	2	4	12	76	28	0	3	49	237

TABLE C- 7 Eclipse Software Metrics V3.5

Eclipse 3.5										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
JDTCompilerAdapter	34	2	10	6	87	37	3	22	0	392
ClasspathDirectory	10	2	4	10	46	26	2	7	24	157
FileFinder	6	2	0	2	18	10	0	4	1	28
CodeFormatter	1	2	9	3	12	3	0	1	3	20
AbstractMethodDeclaration	11	2	17	28	99	32	1	4	216	364
AbstractVariableDeclaration	5	2	16	11	29	14	0	1	53	78
Argument	8	2	1	8	45	11	0	3	28	140
ArrayAllocationExpression	18	2	3	5	48	8	1	5	0	126
ArrayInitializer	18	2	2	5	60	7	0	3	0	178
ClassFileReader	46	3	25	45	213	76	3	21	600	801
ClassFileStruct	5	2	3	9	24	12	0	3	0	71
ClassFormatException	3	4	34	6	28	10	1	4	3	94
FieldInfo	8	3	9	23	99	45	1	14	111	314
InnerClassInfo	4	3	10	5	39	9	0	2	0	82
MethodInfo	17	3	12	26	121	35	0	4	249	404
ConditionalFlowInfo	2	3	2	34	57	34	0	1	0	160
ExceptionHandlingFlowContext	8	3	10	9	162	15	1	5	4	204
FlowInfo	5	2	8	41	25	41	0	1	816	172
InitializationFlowContext	2	4	5	3	164	7	1	5	0	74
LabelFlowContext	3	4	1	2	134	4	0	1	0	30

Eclipse 3.5										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
LoopingFlowContext	19	4	16	11	231	19	1	5	38	550
BooleanConstant	1	3	3	5	926	6	0	1	4	24
ByteConstant	1	3	1	11	926	12	0	1	0	40
CompilerOptions	173	2	228	16	367	25	2	8	16	1332
Constant	5	2	1	34	925	37	0	3	219	1365
IntConstant	1	3	17	11	942	12	0	2	0	74
LongConstant	3	3	3	11	928	12	0	2	0	47
StringConstant	1	3	1	4	926	4	1	1	4	19
RecoveredBlock	22	4	9	20	143	25	1	11	35	324
RecoveredField	18	3	8	11	116	15	1	7	0	234
RecoveredImport	2	3	1	6	77	6	0	1	0	31
RecoveredInitializer	9	4	7	13	130	17	2	8	6	249
RecoveredMethod	23	3	13	20	178	27	2	9	0	514
BufferedContent	3	2	2	8	24	8	0	1	4	53
CompareConfiguration	4	2	21	33	54	52	4	8	404	341
CompareUI	2	2	9	24	39	25	1	4	276	113
CompareViewerPane	3	2	5	20	41	20	0	2	168	170
CompareViewerSwitchingPane	10	3	4	14	82	14	1	3	23	200
HistoryItem	3	2	2	7	20	7	0	2	3	47
NavigationAction	6	2	2	2	18	3	0	2	0	47

Eclipse 3.5										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
ResourceNode	5	3	2	16	52	19	1	3	0	119
AddFromHistoryAction	8	2	1	3	24	7	1	5	6	104
BinaryCompareViewer	12	2	7	4	27	6	2	4	2	105
ChangePropertyAction	3	2	4	8	24	8	2	2	6	57
CompareAction	3	2	3	3	17	3	0	0	0	34
CompareMessages	1	2	113	0	11	0	1	1	0	115
DocLineComparator	7	2	5	7	28	11	0	2	5	110
DocumentManager	6	2	3	4	26	9	1	3	0	42
ExceptionHandler	3	2	1	8	24	12	0	5	26	62
ImageMergeViewer	7	2	6	7	26	9	1	4	5	100
ResizableDialog	10	2	9	6	30	6	2	3	7	122
SimpleTextViewer	3	2	2	4	18	4	0	1	2	42
DiffContainer	4	3	2	6	31	7	1	2	0	49
DiffElement	2	2	2	6	17	6	0	1	11	31
DiffNode	9	4	6	19	75	22	0	2	92	179
DocumentRangeNode	7	2	8	26	57	33	2	4	161	213
DebugEvent	3	3	21	7	38	10	0	2	5	129
DebugException	1	2	7	0	11	0	0	0	0	15
Launch	9	2	7	37	107	49	3	10	308	333
Context	8	2	5	10	38	18	1	4	35	121

Eclipse 3.5										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
HrefUtil	7	2	1	5	33	17	1	2	8	70
Link	1	2	2	2	13	2	1	2	0	21
Toc	7	2	10	16	38	20	2	3	104	154
TocFile	2	2	6	7	19	7	1	3	15	47
TocFileParser	3	3	1	1	31	2	0	6	0	36
TocManager	6	2	9	16	57	34	2	15	26	261
URLCoder	3	2	0	4	21	10	0	6	6	66
OverlayIcon	6	2	5	6	34	6	0	0	3	95
PDERuntimePlugin	5	2	6	14	33	16	1	4	43	108
PDERuntimePluginImages	2	2	58	5	17	6	1	3	4	97
RegistryBrowserLabelProvider	22	2	25	5	82	8	0	4	6	325
Color	4	2	1	9	29	11	0	2	0	78
Cursor	29	2	2	5	37	9	0	3	0	395
Device	20	2	35	20	113	36	4	10	148	405
Font	14	2	4	6	46	13	0	4	0	208
FontData	11	2	7	11	38	28	1	5	4	164
FontMetrics	3	2	5	8	19	8	0	0	0	41
Region	5	2	1	24	75	25	0	2	0	177
Canvas	12	2	2	7	78	22	1	2	1	207
Caret	9	2	11	17	76	30	2	2	0	258

Eclipse 3.5										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
ColorDialog	6	2	1	3	19	3	0	0	0	58
DirectoryDialog	14	2	2	5	30	7	1	1	0	109
FileDialog	33	2	8	14	79	26	1	2	76	329
Group	3	2	1	5	27	12	1	1	39	85
Label	7	2	3	7	60	18	1	1	92	225
MessageBox	23	2	1	3	60	7	1	1	4	160
ProgressBar	4	2	0	9	31	12	0	0	66	79
Sash	13	2	9	3	80	20	2	4	128	295
Scale	4	2	2	13	47	18	0	1	132	138
Scrollable	8	2	3	5	86	27	0	0	59	278
TabItem	11	2	4	9	48	16	1	1	33	152
ToolBar	7	2	2	9	92	32	0	3	100	324

TABLE C- 8 Eclipse Software Metrics V3.6

Eclipse 3.6										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
JDTCompilerAdapter	34	2	10	6	87	37	3	22	0	392
ClasspathDirectory	10	2	4	10	46	27	2	6	24	157
FileFinder	6	2	0	2	18	10	0	4	1	28
CodeFormatter	1	2	9	3	12	3	0	1	3	20
AbstractMethodDeclaration	11	2	17	28	99	32	1	5	216	364
AbstractVariableDeclaration	5	2	16	11	29	14	0	1	53	78
Argument	10	2	1	8	48	11	0	3	28	140
ArrayAllocationExpression	18	2	3	5	48	8	1	5	0	126
ArrayInitializer	18	2	2	5	60	7	0	3	0	178
ClassFileReader	46	3	25	45	213	76	3	21	600	801
ClassFileStruct	5	2	3	9	24	12	0	3	0	71
ClassFormatException	3	4	34	6	28	10	1	4	3	94
FieldInfo	8	3	9	23	99	45	1	14	111	314
InnerClassInfo	4	3	10	5	39	9	0	2	0	82
MethodInfo	17	3	12	26	121	35	0	4	249	404
ConditionalFlowInfo	2	3	2	36	66	36	0	1	0	160
ExceptionHandlingFlowContext	8	3	10	9	177	15	1	5	4	204
FlowInfo	7	2	8	44	32	44	0	1	936	172
InitializationFlowContext	2	4	5	3	179	7	1	5	0	74
LabelFlowContext	3	4	1	2	149	4	0	1	0	30

Eclipse 3.6										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
LoopingFlowContext	41	4	16	11	295	22	1	5	34	550
BooleanConstant	1	3	3	5	926	6	0	1	4	24
ByteConstant	1	3	1	11	926	12	0	1	0	40
CompilerOptions	181	2	236	15	374	23	1	7	15	1332
Constant	5	2	1	34	925	37	0	3	219	1365
IntConstant	1	3	17	11	942	12	0	2	0	74
LongConstant	3	3	3	11	928	12	0	2	0	47
StringConstant	1	3	1	4	926	4	1	1	4	19
RecoveredBlock	22	4	9	20	143	25	1	11	35	324
RecoveredField	18	3	8	12	123	16	1	7	0	234
RecoveredImport	2	3	1	6	77	6	0	1	0	31
RecoveredInitializer	9	4	7	13	134	17	2	8	6	249
RecoveredMethod	23	3	13	20	178	27	2	9	0	514
BufferedContent	3	2	2	8	24	8	0	1	4	53
CompareConfiguration	4	2	21	33	54	52	4	8	404	341
CompareUI	2	2	9	24	39	25	1	4	276	113
CompareViewerPane	3	2	5	20	41	20	0	2	168	170
CompareViewerSwitchingPane	11	3	4	14	83	14	1	2	23	200
HistoryItem	3	2	2	7	20	7	0	2	3	47
NavigationAction	6	2	2	2	18	3	0	2	0	47

Eclipse 3.6										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
ResourceNode	5	3	2	16	52	19	1	3	0	119
AddFromHistoryAction	8	2	1	3	24	7	1	5	6	104
BinaryCompareViewer	12	2	7	4	27	6	2	4	2	105
ChangePropertyAction	3	2	4	8	24	8	2	2	6	57
CompareAction	3	2	3	3	17	3	0	0	0	34
CompareMessages	7	2	5	7	28	11	0	2	5	115
DocLineComparator	6	2	3	4	26	9	1	3	0	110
DocumentManager	3	2	1	8	24	12	0	5	26	42
ExceptionHandler	7	2	6	7	26	9	1	4	5	62
ImageMergeViewer	10	2	9	6	30	6	2	3	7	100
ResizableDialog	3	2	2	4	18	4	0	1	2	122
SimpleTextViewer	4	3	2	6	31	7	1	2	0	42
DiffContainer	2	2	2	6	17	6	0	1	11	49
DiffElement	9	4	6	19	75	22	0	2	92	31
DiffNode	7	2	8	26	57	33	2	4	161	179
DocumentRangeNode	3	3	21	7	38	10	0	2	5	213
DebugEvent	1	2	7	0	11	0	0	0	0	129
DebugException	9	2	7	37	107	49	3	10	308	15
Launch	8	2	5	10	38	18	1	4	35	333
Context	7	2	1	5	33	17	1	2	8	121

Eclipse 3.6										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
HrefUtil	1	2	2	2	13	2	1	2	0	70
Link	7	2	11	18	40	22	2	3	137	21
Toc	2	2	6	7	19	7	1	3	15	154
TocFile	3	3	1	1	31	3	0	6	0	47
TocFileParser	6	2	9	16	59	35	2	16	20	36
TocManager	3	2	0	6	24	12	0	6	15	261
URLCoder	6	2	5	6	34	6	0	0	3	66
OverlayIcon	5	2	6	14	33	16	1	4	43	95
PDERuntimePlugin	2	2	58	5	17	6	1	3	4	108
PDERuntimePluginImages	22	2	25	5	82	8	0	4	6	97
RegistryBrowserLabelProvider	4	2	1	9	29	11	0	2	0	325
Color	29	2	2	5	37	9	0	3	0	78
Cursor	20	2	35	20	113	36	4	10	148	395
Device	14	2	4	6	46	13	0	4	0	405
Font	11	2	7	11	38	28	1	5	4	208
FontData	3	2	5	8	19	8	0	0	0	164
FontMetrics	5	2	1	24	75	25	0	2	0	41
Region	12	2	2	7	78	23	1	2	0	177
Canvas	9	2	11	17	76	30	2	2	0	207
Caret	6	2	1	3	19	3	0	0	0	258

Eclipse 3.6										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
ColorDialog	1	2	106	0	11	0	1	1	0	58
DirectoryDialog	14	2	2	5	30	7	1	1	0	109
FileDialog	40	2	8	14	86	28	1	2	76	329
Group	3	2	1	5	27	12	1	1	39	85
Label	9	2	3	7	63	18	1	1	92	225
MessageBox	23	2	1	3	60	7	1	1	4	160
ProgressBar	4	2	0	9	31	12	0	0	66	79
Sash	13	2	9	3	80	20	2	4	128	295
Scale	4	2	2	13	47	18	0	1	132	138
Scrollable	8	2	3	5	89	28	0	0	62	278
TabItem	11	2	4	9	48	16	1	1	33	152
ToolBar	7	2	2	9	99	34	0	3	125	324

TABLE C- 9 NetBeans Software Metrics V3.5.1

NetBeans 3.5.1										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
IDESettingsBeanInfo	2	3	0	2	21	6	0	7	1	84
NbPlaces	4	2	2	19	36	26	1	8	184	117
NonGuiMain	28	3	11	12	112	39	3	22	60	364
Plain	3	2	2	7	21	14	0	8	19	65
WarmUpSupport	4	2	3	1	16	2	1	2	1	40
AboutAction	1	2	1	4	15	4	0	1	6	20
ConfigureShortcutsAction	1	2	0	4	15	4	0	1	6	31
GlobalPropertiesAction	1	2	1	5	16	5	0	1	10	28
HTMLViewAction	10	2	1	4	24	4	0	2	6	73
SystemExit	1	2	1	4	15	4	0	1	6	24
ViewRuntimeTabAction	1	2	0	4	15	4	0	2	6	26
DefaultParser	6	3	6	14	43	21	1	9	63	107
XMLEnvironmentProvider	2	2	2	3	16	3	1	1	3	36
PerformanceEvent	2	3	2	5	17	6	1	2	0	33
FileStateEditor	4	2	4	4	20	6	1	6	0	60
XML	2	2	1	2	15	2	0	0	0	19
FormatterIndentEngine	4	2	6	16	34	23	2	14	0	107
NbEditorUtilities	4	2	0	10	31	16	0	10	45	129
AnnotationTypesFolder	13	2	4	3	31	6	2	8	3	131
FontsColorsMIMEOptionFile	31	2	12	2	51	24	1	12	0	182

NetBeans 3.5.1										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
MIMEOptionFolder	13	2	6	6	50	16	2	13	0	265
OptionUtilities	7	2	1	24	89	64	1	30	276	368
RADVisualContainer	5	2	6	17	54	29	2	14	76	187
EventCustomEditor	8	6	10	7	378	26	5	13	0	241
FormDataLoader	5	2	2	6	26	9	1	2	13	85
MethodPicker	5	6	13	9	379	37	6	20	12	195
PersistenceManager	3	2	2	9	22	18	1	8	27	68
PropertyPicker	5	6	15	8	375	38	6	18	8	189
RADContainer	2	2	1	7	21	10	1	2	0	38
AddAction	4	2	2	6	22	8	1	3	13	88
CustomizeLayoutAction	4	2	1	7	24	8	1	3	21	52
EditContainerAction	3	2	2	4	19	4	2	2	6	45
EditFormAction	3	2	1	4	20	4	1	1	6	45
InPlaceEditAction	3	2	1	4	20	4	1	1	6	38
InstallBeanAction	2	2	1	4	16	4	1	1	6	22
InstallToPaletteAction	4	2	1	7	23	7	1	3	21	53
ReloadAction	2	2	2	5	19	7	1	2	4	34
TestAction	5	2	2	6	24	14	1	8	9	94
CustomCodeEditor	1	6	3	6	360	18	3	14	0	95
EnumEditor	7	3	3	6	42	9	1	2	3	110

NetBeans 3.5.1										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
StringArrayEditor	8	2	7	17	42	39	2	17	28	127
UnknownLayoutSupport	1	2	0	2	13	2	0	1	1	11
GridLayoutSupport	6	2	0	4	26	16	0	12	6	106
JToolBarSupport	6	2	0	3	22	10	0	10	3	72
NullLayoutSupport	6	2	3	9	34	15	2	14	30	131
ScrollPaneSupport	4	2	1	6	22	12	1	10	15	63
PaletteItem	5	2	6	15	43	23	2	21	0	167
ScrollPopupMenu	7	6	5	6	411	18	2	10	0	100
SearchPanel	6	6	10	13	372	45	4	18	32	199
SearchTask	2	2	4	2	15	2	1	1	0	46
TextDetail	15	2	10	10	36	11	1	3	25	112
AntProjectDataLoader	5	2	4	7	26	7	1	4	19	114
ShortcutIterator	5	2	17	19	52	30	2	18	27	211
EventSetPatternNode	3	2	0	12	29	14	0	1	78	153
IdxPropertyPatternNode	1	3	0	8	40	9	0	1	36	118
Pattern	2	2	2	7	19	15	1	6	30	37
PropertyActionSettings	8	2	18	19	44	21	1	3	87	114
PropertyPatternNode	3	2	0	12	37	15	0	1	105	184
ClassElementFinder	14	2	2	7	52	18	0	8	1	181
DocumentModelBuilder	6	2	3	18	36	19	0	1	79	127

NetBeans 3.5.1										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
JavaModule	2	2	2	3	16	3	0	1	0	45
NodeFactoryPool	5	2	6	2	27	11	1	6	0	97
ParserAnnotation	7	2	9	9	37	15	1	2	26	110
ParserMessage	0	2	2	4	11	4	0	1	6	12
ContextDisplay	4	2	2	3	21	5	1	6	0	174
CookieDisplay	10	2	2	1	21	5	1	7	0	127
DispatchData	8	2	10	32	58	38	2	9	328	411
DisplayTable	3	6	18	12	517	21	1	13	54	191
EditPanel	11	6	26	9	386	26	4	23	17	291
RequestData	16	2	9	39	117	56	2	13	564	431
TransactionNode	5	2	7	17	37	26	1	7	60	176
BundleNodeCustomizer	2	6	8	10	368	29	5	18	0	208
KeyNode	4	2	3	16	38	26	2	8	78	217
LocaleNodeCustomizer	4	6	10	11	375	37	5	21	36	277
PresentableFileEntry	4	2	7	25	43	30	2	7	280	165
PropertiesStructure	7	2	3	10	39	24	1	7	0	167
PropertyBundleEvent	1	3	8	4	19	5	1	3	0	54
PropertyBundleSupport	3	2	3	7	20	8	1	3	0	40
UtilConvert	20	2	5	17	148	37	1	13	106	352
AutoupdateType	3	2	4	11	25	19	2	8	51	71

NetBeans 3.5.1										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
CertificateDialog	2	6	12	3	360	16	2	13	0	129
SetupPanel	3	6	7	5	363	15	3	11	0	124
Access	14	2	12	6	30	8	0	2	0	62
ClassName	5	2	8	14	42	32	2	4	0	134
Code	5	2	7	9	25	14	0	5	24	74
ConstantPoolReader	7	4	3	15	61	22	0	6	134	159
CPInterfaceMethodInfo	1	2	0	1	12	1	0	0	0	9
Field	5	2	8	13	31	27	1	9	49	97
Method	3	3	3	4	40	10	1	5	7	58

TABLE C- 10 NetBeans Software Metrics V4.0

NetBeans 4.0										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
IDESettingsBeanInfo	2	3	0	2	21	6	0	6	1	70
NbPlaces	4	2	2	19	39	26	1	9	184	138
NonGuiMain	28	2	11	12	67	39	3	22	60	365
Plain	3	2	2	7	21	13	0	7	19	62
WarmUpSupport	9	2	4	1	21	5	1	5	1	60
AboutAction	1	2	0	5	16	5	0	1	10	27
ConfigureShortcutsAction	1	2	0	5	16	5	0	1	10	37
GlobalPropertiesAction	1	2	0	5	16	5	0	1	10	26
HTMLViewAction	10	2	1	5	25	5	1	2	10	78
SystemExit	1	2	1	5	16	5	0	1	10	24
ViewRuntimeTabAction	1	3	0	2	22	2	0	2	1	27
DefaultParser	5	3	6	14	43	23	1	10	63	111
XMLEnvironmentProvider	2	2	2	3	16	3	1	1	3	36
PerformanceEvent	2	3	2	5	17	6	1	2	0	33
FileStateEditor	5	2	4	4	21	6	1	7	0	70
XML	2	2	1	2	15	2	0	0	0	19
FormatterIndentEngine	4	2	6	17	36	24	2	14	0	112
NbEditorUtilities	4	2	0	12	38	18	0	10	66	153
AnnotationTypesFolder	14	2	4	3	32	7	2	9	3	142
FontsColorsMIMEOptionFile	31	2	12	2	51	24	1	12	0	182

NetBeans 4.0										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
MIMEOptionFolder	13	2	6	6	49	16	2	13	0	266
OptionUtilities	8	2	1	28	94	68	1	31	378	383
RADVisualContainer	5	2	6	17	54	29	2	14	76	186
EventCustomEditor	8	6	10	7	378	26	5	13	0	239
FormDataLoader	2	2	2	5	18	6	1	2	8	54
MethodPicker	5	6	13	9	379	37	6	20	12	192
PersistenceManager	3	2	2	9	22	18	1	8	27	69
PropertyPicker	5	6	15	8	375	38	6	18	8	186
RADContainer	2	2	1	7	21	10	1	2	0	38
AddAction	4	2	1	8	23	10	1	3	22	90
CustomizeLayoutAction	4	2	1	8	26	9	1	3	28	57
EditContainerAction	4	2	2	5	21	5	2	2	10	51
EditFormAction	4	2	1	5	22	5	1	1	10	51
InPlaceEditAction	4	2	1	5	23	5	1	1	10	49
InstallBeanAction	2	2	1	4	16	4	1	1	6	22
InstallToPaletteAction	2	2	1	6	18	6	1	2	15	30
ReloadAction	2	2	1	4	17	4	1	1	6	31
TestAction	5	2	2	7	25	15	1	8	15	97
CustomCodeEditor	1	6	3	6	360	18	3	14	0	95
EnumEditor	7	3	3	6	42	9	1	2	3	118

NetBeans 4.0										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
StringArrayEditor	8	2	7	17	42	39	2	17	28	127
UnknownLayoutSupport	1	2	0	2	13	2	0	1	1	11
GridLayoutSupport	6	2	0	4	26	16	0	12	6	105
JToolBarSupport	6	2	0	3	22	10	0	10	3	72
NullLayoutSupport	6	2	3	9	34	15	2	14	30	130
ScrollPaneSupport	7	6	5	7	413	24	2	18	0	73
PaletteItem	11	2	11	16	49	29	3	18	0	167
ScrollPopupMenu	5	2	0	6	23	18	0	12	15	116
SearchPanel	6	6	11	15	381	38	5	17	10	220
SearchTask	3	2	9	3	25	8	2	3	0	86
TextDetail	11	2	9	11	34	14	1	5	33	118
AntProjectDataLoader	2	2	5	5	17	5	1	2	10	57
ShortcutIterator	4	2	4	12	42	23	1	11	35	141
EventSetPatternNode	3	2	0	11	28	13	0	1	66	150
IdxPropertyPatternNode	1	3	0	7	39	8	0	1	28	115
Pattern	2	2	2	7	19	15	1	6	30	37
PropertyActionSettings	2	2	6	7	20	7	1	1	15	42
PropertyPatternNode	3	2	0	11	36	14	0	1	91	181
ClassElementFinder	15	2	2	9	58	22	0	9	30	214
DocumentModelBuilder	1	2	3	17	28	17	0	1	136	58

NetBeans 4.0										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
JavaModule	2	2	2	3	15	3	0	1	3	38
NodeFactoryPool	7	2	5	3	35	12	2	8	6	129
ParserAnnotation	7	2	10	9	40	17	1	3	37	126
ParserMessage	0	2	2	4	11	4	0	1	6	12
ContextDisplay	4	2	2	3	21	5	1	6	0	174
CookieDisplay	10	2	1	1	21	6	0	10	0	150
DispatchData	8	2	10	32	58	38	2	9	328	411
DisplayTable	3	6	19	14	518	24	1	16	77	212
EditPanel	8	6	24	5	385	24	4	23	10	264
RequestData	16	2	9	39	121	57	2	13	560	434
TransactionNode	5	2	7	16	36	25	1	7	44	167
BundleNodeCustomizer	2	6	8	10	368	29	5	18	0	207
KeyNode	4	2	3	16	38	24	2	8	78	212
LocaleNodeCustomizer	4	6	10	11	375	37	5	21	36	277
PresentableFileEntry	4	2	7	25	41	30	2	7	280	151
PropertiesStructure	7	2	3	10	39	24	1	7	0	200
PropertyBundleEvent	1	3	8	4	20	7	1	4	0	64
PropertyBundleSupport	3	2	3	7	20	8	1	3	0	44
UtilConvert	10	2	5	13	68	18	1	2	72	133
AutoupdateType	3	2	5	11	26	20	2	8	51	76

NetBeans 4.0										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
CertificateDialog	2	6	12	3	360	16	2	13	0	128
SetupPanel	3	6	7	5	363	16	3	11	0	126
Access	13	2	18	6	29	8	0	2	0	66
ClassName	7	2	8	16	46	37	2	6	2	146
Code	9	2	8	11	30	21	0	6	29	95
ConstantPoolReader	7	4	3	15	61	22	0	6	134	159
CPInterfaceMethodInfo	1	2	0	1	12	1	0	0	0	9
Field	8	2	9	18	41	36	2	12	30	143
Method	7	3	4	13	62	22	1	10	83	134

TABLE C- 11 NetBeans Software Metrics V5.0

NetBeans 5.0										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
IDESettingsBeanInfo	2	3	0	2	21	6	0	6	1	80
NbPlaces	4	2	2	11	28	18	1	8	60	97
NonGuiMain	28	2	11	12	67	39	3	22	60	366
Plain	3	2	2	7	21	13	0	7	19	62
WarmUpSupport	9	2	4	1	21	5	1	5	1	60
AboutAction	1	2	0	4	15	4	0	1	6	30
ConfigureShortcutsAction	1	2	0	5	16	5	0	1	10	23
GlobalPropertiesAction	1	2	0	5	16	5	0	1	10	26
HTMLViewAction	10	2	1	5	25	5	1	2	10	78
SystemExit	1	2	1	6	17	6	0	1	15	28
ViewRuntimeTabAction	1	3	0	2	22	2	0	1	1	26
DefaultParser	5	3	6	14	43	23	1	10	63	111
XMLEnvironmentProvider	2	2	2	3	16	3	1	1	3	36
PerformanceEvent	2	3	2	5	17	6	1	2	0	33
FileStateEditor	5	2	4	4	21	6	1	7	0	70
XML	2	2	1	2	15	2	0	0	0	19
FormatterIndentEngine	4	2	6	17	36	24	2	14	0	115
NbEditorUtilities	4	2	0	14	44	24	0	12	91	182
AnnotationTypesFolder	15	2	4	3	33	7	2	9	3	150
FontsColorsMIMEOptionFile	32	2	12	2	52	24	1	12	0	190

NetBeans 5.0										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
MIMEOptionFolder	13	2	6	6	50	16	2	13	0	276
OptionUtilities	8	2	1	28	94	68	1	31	378	382
RADVisualContainer	5	2	6	17	64	32	2	14	109	239
EventCustomEditor	8	6	10	7	378	26	5	13	0	239
FormDataLoader	2	2	2	5	18	6	1	2	8	46
MethodPicker	8	6	12	11	383	42	6	24	39	236
PersistenceManager	3	2	2	9	22	18	1	8	27	69
PropertyPicker	8	6	14	10	379	45	6	24	33	227
RADContainer	2	2	1	7	21	10	1	2	0	38
AddAction	4	2	1	8	23	10	1	3	22	90
CustomizeLayoutAction	4	2	1	7	25	8	1	3	21	54
EditContainerAction	4	2	2	5	21	5	2	2	10	50
EditFormAction	4	2	1	5	22	5	1	1	10	50
InPlaceEditAction	4	2	1	5	23	5	1	1	10	47
InstallBeanAction	2	2	1	4	16	4	1	1	6	25
InstallToPaletteAction	2	2	1	6	18	6	1	2	15	33
ReloadAction	2	2	1	4	17	4	1	1	6	31
TestAction	7	2	2	7	27	14	1	9	15	117
CustomCodeEditor	1	6	4	6	360	17	3	14	0	96
EnumEditor	7	3	3	6	42	9	1	2	3	118

NetBeans 5.0										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
StringArrayEditor	8	2	7	17	54	43	2	17	28	156
UnknownLayoutSupport	1	2	0	2	13	2	0	1	1	9
GridLayoutSupport	8	2	0	4	29	16	0	12	6	118
JToolBarSupport	7	2	0	4	26	13	0	12	6	87
NullLayoutSupport	6	2	3	9	34	15	2	14	30	130
ScrollPaneSupport	5	2	0	6	23	18	0	12	15	73
PaletteItem	11	2	11	16	49	29	3	19	0	171
ScrollPopupMenu	7	6	5	7	415	24	2	18	0	122
SearchPanel	8	6	11	17	388	40	5	17	0	261
SearchTask	3	2	9	4	26	9	2	3	0	98
TextDetail	11	2	10	11	34	14	1	5	33	123
AntProjectDataLoader	2	2	5	5	17	5	1	2	8	37
ShortcutIterator	4	2	4	12	42	23	1	11	35	141
EventSetPatternNode	3	2	0	9	26	11	0	1	45	129
IdxPropertyPatternNode	1	3	0	7	38	8	0	1	28	102
Pattern	2	2	3	7	20	15	1	6	30	49
PropertyActionSettings	2	2	6	6	19	6	1	1	9	39
PropertyPatternNode	3	2	0	10	35	13	0	1	78	171
ClassElementFinder	15	2	2	9	58	22	0	9	30	214
DocumentModelBuilder	1	2	3	17	28	17	0	1	136	58

NetBeans 5.0										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
JavaModule	1	2	0	2	13	2	0	0	1	13
NodeFactoryPool	7	2	5	3	35	12	2	8	6	129
ParserAnnotation	7	2	10	9	40	17	1	3	37	126
ParserMessage	0	2	2	4	11	4	0	1	6	12
ContextDisplay	4	2	2	3	21	5	1	6	0	174
CookieDisplay	10	2	1	1	21	6	0	10	0	150
DispatchData	8	2	10	32	58	38	2	9	328	411
DisplayTable	3	6	19	14	518	24	1	16	77	212
EditPanel	8	6	24	5	385	24	4	22	10	257
RequestData	16	2	9	39	121	57	2	13	560	434
TransactionNode	12	2	23	18	47	27	2	8	55	232
BundleNodeCustomizer	2	6	8	10	368	29	5	18	0	207
KeyNode	4	2	3	16	38	24	2	8	78	212
LocaleNodeCustomizer	4	6	10	11	375	37	5	21	36	277
PresentableFileEntry	4	2	7	25	41	30	2	7	280	151
PropertiesStructure	7	2	3	10	39	24	1	7	0	200
PropertyBundleEvent	1	3	8	4	20	7	1	4	0	64
PropertyBundleSupport	3	2	3	7	20	8	1	3	0	44
UtilConvert	10	2	5	13	68	18	1	2	72	133
AutoupdateType	3	2	5	11	26	20	2	8	51	78

NetBeans 5.0										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
CertificateDialog	2	6	12	3	360	16	2	14	0	128
SetupPanel	3	6	7	5	363	16	3	11	0	126
Access	13	2	18	6	29	8	0	2	0	66
ClassName	7	2	8	16	46	37	2	6	2	146
Code	13	2	9	12	35	23	0	7	38	109
ConstantPoolReader	7	4	3	15	61	22	0	6	134	159
CPInterfaceMethodInfo	1	2	0	1	12	1	0	0	0	9
Field	8	2	9	18	41	36	2	12	30	143
Method	7	3	4	13	62	22	1	10	83	134

TABLE C- 12 NetBeans Software Metrics V5.5.1

NetBeans 5.5.1										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
IDESettingsBeanInfo	2	3	0	2	21	6	0	8	1	86
NbPlaces	4	2	2	11	28	18	1	8	60	97
NonGuiMain	28	2	11	12	67	39	3	22	60	366
Plain	3	2	2	7	21	13	0	7	19	62
WarmUpSupport	9	2	4	1	21	5	1	5	1	60
AboutAction	7	2	0	5	24	16	0	13	15	113
ConfigureShortcutsAction	1	2	0	5	16	5	0	1	10	23
GlobalPropertiesAction	1	2	0	5	16	5	0	1	10	26
HTMLViewAction	10	2	1	5	25	5	1	2	10	78
SystemExit	1	2	1	6	17	6	0	1	15	28
ViewRuntimeTabAction	1	3	0	2	22	2	0	1	1	26
DefaultParser	5	3	6	14	43	23	1	10	63	111
XMLEnvironmentProvider	2	2	2	3	16	3	1	1	3	36
PerformanceEvent	2	3	2	5	17	6	1	2	0	33
FileStateEditor	5	2	4	4	21	6	1	7	0	70
XML	2	2	1	2	15	2	0	0	0	19
FormatterIndentEngine	4	2	6	17	36	24	2	14	0	115
NbEditorUtilities	4	2	0	14	44	24	0	12	91	182
AnnotationTypesFolder	15	2	4	3	33	7	2	9	3	150
FontsColorsMIMEOptionFile	32	2	12	2	52	24	1	12	0	190

NetBeans 5.5.1										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
MIMEOptionFolder	13	2	6	6	50	16	2	13	0	276
OptionUtilities	8	2	1	28	94	68	1	31	378	382
RADVisualContainer	7	2	6	17	72	38	2	17	73	279
EventCustomEditor	8	6	10	7	378	26	5	13	0	239
FormDataLoader	2	2	2	5	18	6	1	2	8	46
MethodPicker	8	6	12	11	383	42	6	24	39	236
PersistenceManager	3	2	2	9	22	18	1	8	27	69
PropertyPicker	8	6	14	10	379	45	6	24	33	227
RADContainer	2	2	1	7	21	10	1	2	0	38
AddAction	4	2	1	8	23	10	1	3	22	90
CustomizeLayoutAction	4	2	1	7	25	8	1	3	21	54
EditContainerAction	4	2	2	5	21	5	2	2	10	50
EditFormAction	4	2	1	5	22	5	1	1	10	50
InPlaceEditAction	4	2	1	5	23	5	1	1	10	47
InstallBeanAction	2	2	1	4	16	4	1	1	6	25
InstallToPaletteAction	2	2	1	6	18	6	1	2	15	33
ReloadAction	2	2	1	4	17	4	1	1	6	31
TestAction	7	2	2	7	27	14	1	9	15	117
CustomCodeEditor	1	6	4	6	360	17	3	14	0	96
EnumEditor	7	3	3	6	42	9	1	2	3	118

NetBeans 5.5.1										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
StringArrayEditor	8	2	7	17	54	43	2	17	28	156
UnknownLayoutSupport	1	2	0	2	13	2	0	1	1	9
GridLayoutSupport	8	2	0	4	29	16	0	12	6	118
JToolBarSupport	7	2	0	4	26	13	0	12	6	87
NullLayoutSupport	6	2	3	9	34	15	2	14	30	130
ScrollPaneSupport	5	2	0	6	23	18	0	12	15	73
PaletteItem	11	2	11	16	49	29	3	19	0	171
ScrollPopupMenu	7	6	5	7	415	24	2	18	0	122
SearchPanel	7	6	5	7	415	24	2	18	0	261
SearchTask	8	6	11	17	388	40	5	17	0	98
TextDetail	3	2	9	4	26	9	2	3	0	123
AntProjectDataLoader	2	2	5	5	17	5	1	2	8	37
ShortcutIterator	4	2	4	12	42	23	1	11	35	141
EventSetPatternNode	3	2	0	9	26	11	0	1	45	129
IdxPropertyPatternNode	1	3	0	7	38	8	0	1	28	102
Pattern	2	2	3	7	20	15	1	6	30	49
PropertyActionSettings	2	2	6	6	19	6	1	1	9	39
PropertyPatternNode	3	2	0	10	35	13	0	1	78	171
ClassElementFinder	15	2	2	9	58	22	0	9	30	214
DocumentModelBuilder	1	2	3	17	28	17	0	1	136	58

NetBeans 5.5.1										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
JavaModule	1	2	0	2	13	2	0	0	1	13
NodeFactoryPool	7	2	5	3	35	12	2	8	6	129
ParserAnnotation	7	2	10	9	40	17	1	3	37	126
ParserMessage	0	2	2	4	11	4	0	1	6	12
ContextDisplay	4	2	2	3	21	5	1	6	0	174
CookieDisplay	10	2	1	1	21	6	0	10	0	150
DispatchData	8	2	10	32	58	38	2	9	328	411
DisplayTable	3	6	19	14	518	24	1	16	77	212
EditPanel	8	6	24	5	385	24	4	22	10	257
RequestData	3	2	0	10	35	13	0	1	78	434
TransactionNode	3	6	7	5	363	16	3	11	0	232
BundleNodeCustomizer	2	6	8	10	368	29	5	18	0	207
KeyNode	4	2	3	16	38	24	2	8	78	212
LocaleNodeCustomizer	4	6	10	11	375	37	5	21	36	277
PresentableFileEntry	4	2	7	25	41	30	2	7	280	151
PropertiesStructure	7	2	3	10	39	24	1	7	0	200
PropertyBundleEvent	1	3	8	4	20	7	1	4	0	64
PropertyBundleSupport	3	2	3	7	20	8	1	3	0	44
UtilConvert	12	2	23	18	47	27	2	8	55	133
AutoupdateType	4	2	5	11	27	20	2	9	51	81

NetBeans 5.5.1										
Class Name	**V(G)**	**DIT**	**NOA**	**NLM**	**WMC**	**RFC**	**DAC**	**CBO**	**LCOM**	**LOC**
CertificateDialog	2	6	12	3	360	16	2	14	0	128
SetupPanel	16	2	9	39	121	57	2	13	560	126
Access	13	2	18	6	29	8	0	2	0	66
ClassName	7	2	8	16	46	37	2	6	2	146
Code	13	2	9	12	35	23	0	7	38	109
ConstantPoolReader	7	4	3	15	61	22	0	6	134	159
CPInterfaceMethodInfo	1	2	0	1	12	1	0	0	0	9
Field	8	2	9	18	42	36	2	12	30	143
Method	7	3	4	13	62	22	1	10	83	134

REFRENCES

[1] S. McConnell and C. Complete, "A Practical Handbook of Software Construction," ed: Microsoft Press Redmond, WA, USA, 1993.

[2] M. Alshayeb, *et al.*, "Towards measuring object-oriented class stability," *Software, IET*, vol. 5, pp. 415-424, 2011.

[3] W. Li, *et al.*, "An empirical study of object-oriented system evolution," *Information and Software Technology*, vol. 42, pp. 373-381, 2000.

[4] D. Ratiu, "Using history information to improve design flaws detection," 2004.

[5] D. Grosser, *et al.*, "An analogy-based approach for predicting design stability of Java classes," 2003.

[6] M. Thwin and T. Quah, "Application of neural networks for software quality prediction using object-oriented metrics," *Journal of Systems and Software*, vol. 76, pp. 147-156, 2005.

[7] S. Bouktif, *et al.*, "Improving rule set based software quality prediction: A genetic algorithm-based approach," *Journal of Object Technology*, vol. 3, pp. 227–241, 2004.

[8] D. Azar, *et al.*, "Predicting stability of classes in an object-oriented system," *Journal of Computational Methods in Science and Engineering*, vol. 10, pp. 39-49, 2010.

[9] M. Elish and D. Rine, "Investigation of metrics for Object-Oriented design logical stability," in *Proceedings of the Seventh European Conference of Software Maintenance and Reengineering*, March 2003, pp. 193-200.

[10] M. Alshayeb and W. Li, "An empirical study of system design instability metric and design evolution in an agile software process," *Journal of Systems and Software*, vol. 74, pp. 269-274, 2005.

[11] M. E. Fayad and A. Altman, "Thinking objectively: an introduction to software stability," *Communications of the ACM,* vol. 44, p. 95, 2001.

[12] M. Fayad, "Accomplishing software stability," *Communications of the ACM,* vol. 45, pp. 111-115, 2002.

[13] M. E. Fayad, "How to deal with software stability," *Communications of the ACM,* vol. 45, pp. 109-112, 2002.

[14] S. S. Yau and J. S. Collofello, "Design stability measures for software maintenance," *IEEE Transactions on Software Engineering,* pp. 849-856, 1985.

[15] N. L. Soong, "A program stability measure," 1977, pp. 163-173.

[16] Y. S. Hassan, "Measuring software architectural stability using retrospective analysis," KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS, 2008.

[17] J. Bansiya, "Evaluating framework architecture structural stability," *ACM Computing Surveys (CSUR),* vol. 32, p. 18, 2000.

[18] M. Ahmed, *et al.*, "Measuring Architectural Stability in Object Oriented Software," *Stable Analysis Patterns: A True Problem Understanding with UML,* p. 21, 2004.

[19] S. A. Tonu, *et al.*, "Evaluating architectural stability using a metric-based approach," 2006.

[20] L. Briand, *et al.*, "On the application of measurement theory in software engineering," *Empirical Software Engineering Journal,* vol. 1, pp. 61–68, 1966.

[21] V. Basili, *et al.*, "Metric Analysis and Validation Across FORTRAN Projects," *IEEE Transactions on Software Engineering,* vol. 9, pp. 652-663, 1983.

[22] T. Khoshgoftaar, *et al.*, "An Information Theory-Based Approach to Quantify the Contribution of a Software Metric," *Journal of Systems and Software,* vol. 36, pp. 103-113, 1997.

[23] B. Kitchenham, *et al.*, "Towards a framework for software measurement validation," *Software Engineering, IEEE Transactions on,* vol. 21, pp. 929-944, 1995.

[24] A. Melton, *Software Measurement*: Thomson Computer Press, 1996.

[25] N. Schneidewind, "Methodology for Validating Software Metrics," *IEEE Transactions on Software Engineering,* vol. 18, pp. 410-422, 1992.

[26] Y. Singh and P. Kumar, "Prediction of Software Reliability Using Feed Forward Neural Networks," 2010, pp. 1-5.

[27] C. Zhong, *et al.*, "Software Quality Prediction Method with Hybrid Applying Principal Components Analysis and Wavelet Neural Network and Genetic Algorithm," *JDCTA: International Journal of Digital Content Technology and its Applications,* vol. 5, pp. 225-234, 2011.

[28] S. Kanmani, *et al.*, "Object-oriented software fault prediction using neural networks," *Information and Software Technology,* vol. 49, pp. 483-492, 2007.

[29] W. Peng, *et al.*, "An approach of software quality prediction based on relationship analysis and prediction model," 2009, pp. 713-717.

[30] H. A. Al-Jamimi and L. Ghouti, "Efficient prediction of software fault proneness modules using support vector machines and probabilistic neural networks," 2011, pp. 251-256.

[31] I. Gondra, "Applying machine learning to software fault-proneness prediction," *Journal of Systems and Software,* vol. 81, pp. 186-195, 2008.

[32] Q. Wang, *et al.*, "Extract rules from software quality prediction model based on neural network," in *Proceedings of The 16th IEEE International Conference on Tools with Artificial Intelligence, ICTAI'04*, 2004.

[33] T. M. Khoshgoftaar, *et al.*, "A comparative study of pattern recognition techniques for quality evaluation of telecommunications software," *Selected Areas in Communications, IEEE Journal on,* vol. 12, pp. 279-291, 1994.

[34] M. J. Khan, *et al.*, "Comparative study of various artificial intelligence techniques to predict software quality," 2006, pp. 173-177.

[35] K. O. Elish and M. O. Elish, "Predicting defect-prone software modules using support vector machines," *Journal of Systems and Software,* vol. 81, pp. 649-660, 2008.

[36] Z. Yan, *et al.*, "Software Defect Prediction Using Fuzzy Support Vector Regression," *Advances in Neural Networks-ISNN 2010*, pp. 17-24, 2010.

[37] Y. Singh, *et al.*, "Software Fault Proneness Prediction Using Support Vector Machines," 2009, pp. 1-3.

[38] F. Xing, *et al.*, "A novel method for early software quality prediction based on support vector machine," in *Proceedings of The 16th IEEE International Symposium on Software Reliability Engineering*, 2005.

[39] B. Twala, "Software faults prediction using multiple classifiers," 2011, pp. 504-510.

[40] Y. Kamei, *et al.*, "Empirical evaluation of svm-based software reliability model," 2006, pp. 39-41.

[41] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*: The MIT Press, 1992.

[42] S. Wagner, "A Bayesian network approach to assess and predict software quality using activity-based quality models," *Information and Software Technology*, vol. 52, pp. 1230-1241, 2010.

[43] Ł. Radliński, "A conceptual Bayesian net model for integrated software quality prediction," 2011, pp. 49-60.

[44] H. Jia, *et al.*, "Predicting Fault-Prone Modules: A Comparative Study," *Software Engineering Approaches for Offshore and Outsourced Development*, pp. 45-59, 2009.

[45] A. Aamodt and E. Plaza, "Case-based reasoning: Foundational issues, methodological variations, and system approaches," *AI communications*, vol. 7, pp. 39-59, 1994.

[46] E. Paikari, *et al.*, "Customization support for CBR-based defect prediction," 2011, p. 16.

[47] D. Gupta, *et al.*, "Analysis of Clustering Techniques for Software Quality Prediction," 2012, pp. 6-9.

[48] A. Kaur, *et al.*, "An empirical approach for software fault prediction," 2010, pp. 261-265.

[49] X. Yuan, *et al.*, "An application of fuzzy clustering to software quality prediction," in *Proceedings of The 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology*, 2000.

[50] D. Steinberg and P. Colla, "CART: classification and regression trees," *San Diego, CA*, 1997.

[51] T. M. Khoshgoftaar, *et al.*, "Classification tree models of software quality over multiple releases," 1999, pp. 116-125.

[52] D. Grosser, *et al.*, "Predicting software stability using case-based reasoning," 2002, pp. 295-298.

[53] R. Martin, "OO design quality metrics," *An analysis of dependencies*, 1994.

[54] Y. Hassan, "Measuring software architectural stability using retrospective analysis," M.S. thesis, King Fahd University of Petroleum & Minerals, Dhahran, 2007.

[55] S. Tonu, *et al.*, "Evaluating Architectural Stability Using a Metric-Based Approach," in *Proceedings of the Conference on Software Maintenance and Reengineering (CSMR'06)*, 2006, pp. 261-270.

[56] R. Bahsoon and W. Emmerich, "Evaluating Software Architectures: Development, Stability, and Evolution," in *Proceedings of ACS/IEEE Int. Conf. on Computer Systems and Applications*, Tunis, Tunisia, 2003, pp. 47-56.

[57] L. C. Briand, *et al.*, "Investigating quality factors in object-oriented designs: an industrial case study," 1999, pp. 345-354.

[58] http://www.dtreg.com/.

[59] J. S. Alghamdi, *et al.*, "OOMeter: A software quality assurance tool," 2005, pp. 190-191.

[60] M. A. Naji, "Measuring Object-Oriented class stability," KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS, 2008.

[61] http://www.android.com/.

[62] http://www.eclipse.org.

[63] www.netbeans.org.

[64] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *Software Engineering, IEEE Transactions on*, vol. 20, pp. 476-493, 1994.

[65] www.minitab.com/.

[66] D. Rat, *et al.*, "Using history information to improve design flaws detection," in *Proceedings of the Eighth European Conference on Software Maintenance and Reengineering (CSMR'04)*, 2004.

VITA

Yagoub Eisa was born in 1974, Taif, Saudi Arabia. He received his Bachelor Degree (BS) in Computer Science from King Abdulaziz University, Jeddah Saudi Arabia in 1998. From 1998 to 2001 Yagoub worked as a system analyst at SABIC, one of the world's five largest petrochemicals manufacturers. Yagoub primary responsibilities include SAP system administration and maintain the access privileges.

In September 2001 Yagoub joined Saudi Aramco, the world largest oil company, as system analyst. Yagoub involved in a variety of different projects including the Big-Bang implementation to migrate from the distributed legacy systems to an integrated application, the Hydrocarbon Supply Chain Management to integrate different business processes, and different upgrade projects include SAP R/3, SAP Enterprise Portal, and SAP Web Application. Yagoub also has given many responsibilities including customer support, access control, and SAP security administration. Yagoub is a SAP certified technology consultant.

Yagoub joined Information & Computer Science Department at King Fahd University of Petroleum & Minerals in February 2010 and received a Master of Science degree (MS) in Computer Science in May 2012.

Yagoub Eisa can be contacted at eisa1974@yahoo.com